Computing Science

On the Equivalence between Logic Programming Semantics and Argumentation Semantics

Martin Caminada
Samy Sá
João Alcântara

Technical Report Series **ABDN–CS–13–01**

Department of Computing Science **March 2013**
University of Aberdeen
King's College
Aberdeen AB24 3UE

On the Equivalence between Logic Programming Semantics and Argumentation Semantics

Martin Caminada
Samy Sá
João Alcântara

Technical Report ABDN–CS–13–01
Department of Computing Science
University of Aberdeen

March 18, 2013

Abstract: In the current paper, we re-examine the connection between formal argumentation and logic programming from the perspective of semantics. We observe that one particular translation from logic programs to instantiated argumentation (the one described by Wu, Caminada and Gabbay) is able to serve as a basis for describing various equivalences between logic programming semantics and argumentation semantics. In particular, we are able to provide a formal connection between regular semantics for logic programming and preferred semantics for formal argumentation. We also show that there exist logic programming semantics (L-stable semantics) that cannot be captured by any abstract argumentation semantics.

Keywords: formal argumentation theory, logic programming

1 Introduction

The connection between logic programming and formal argumentation goes back to logic programming inspired formalisms like [12] or [14], as well as to the seminal work of Dung [7] in which various connections were pointed out. To some extent, the work of [7] can be seen as an attempt to provide an abstraction of certain aspects of logic programming. The connection between logic programming and argumentation is especially clear when it comes to comparing the different semantics that have been defined for logic programming with the different semantics that have been defined for formal argumentation. In the current paper, we continue this line of research. We do this by pointing out that one particular translation from logic programming to formal argumentation (the one of [17]) is able to account for a whole range of equivalences between logic programming semantics and formal argumentation semantics. This includes both existing results like the equivalence between stable model semantics (LP) and stable semantics (argumentation) [7], between well-founded semantics (LP) and grounded semantics (argumentation) [7], and between partial stable model semantics (LP) and complete semantics [17], as well as a newly proved equivalence between regular model semantics (LP) and preferred semantics (argumentation).

The results of the current paper, however, are relevant for more than just the connection between logic programming and formal argumentation. They also shed light on specific aspects of instantiated argumentation theory in general (e.g. [3, 11, 10]). In particular we show the connection between argument-labellings at the abstract level and conclusion-labellings at the instantiated level. With one notable exception, we are able to show that maximizing (or minimizing) a particular label (`in`, `out` or `undec`) at the argument level coincides with maximizing (or minimizing) the same label at the conclusion level. These results are relevant as they indicate the possibilities (and limitations) of applying argument-based abstractions to formalisms for non-monotonic reasoning.

This paper is structured as follows. First, in Section 2, we will introduce the main concepts to be applied in the current paper, such as the various semantics of abstract argumentation and logic programming upon which we will discuss. Then, in Section 3 we provide an overview of the three step process of instantiated argumentation and how it is applied in the particular context of logic programming based argumentation. In Section 4 we examine some existing work on the minimization and maximization of argument labellings. Similarly, in Section 5 we examine the issue of minimization and maximization of conclusion labellings. The connection between argument labellings and conclusion labellings is then studied in Section 6. Using this connection, we are able to apply it for making the connection between argumentation semantics and logic programming semantics in Section 7. For this, we point out that argument labellings coincide with argument extensions and conclusion labellings coincide with logic programming models. One notable exception on the coincidence between argumentation semantics and logic programming semantics is studied in Section 8, where we examine possible ways in which this coincidence can be restored. A reverse translation from argumentation frameworks to logic programs is then specified in Section 9, and it is observed that for this translation the coincidence of argumentation semantics and logic programming semantics is even stronger than for the translation of (unrestricted) logic programs to argumentation frameworks. Finally, we then round off with a discussion of the obtained results in Section 10.

2 Preliminaries

In this section, we introduce the main definitions used throughout the paper and the first connections between formal argumentation and logic programming. We will start with the definitions of abstract argumentation frameworks and their various semantics and then move on to logic programs and their various semantics, in order to point out the similarities between these concepts.

2.1 Abstract Argumentation Frameworks and Semantics

In the current paper, we follow the approach of Dung [7]. To simplify things, we will restrict ourselves to finite argumentation frameworks.

Definition 1 ([7])**.** *An* argumentation framework *is a pair* (Ar, att) *where* Ar *is a finite set of arguments and* $att \subseteq Ar \times Ar$.

Arguments are related to others by the attack relation att, in the sense that an argument A attacks the argument B iff $(A, B) \in att$. An argumentation framework can be depicted as a directed graph where the arguments are nodes and each attack is an arrow.

Definition 2 ([7])**.** *(defense/conflict-free). Let* (Ar, att) *be an argumentation framework,* $A \in Ar$ *and* $\mathcal{A}rgs \subseteq Ar$*.* $\mathcal{A}rgs$ *is said to be* conflict-free *iff there exists no arguments* $A, B \in \mathcal{A}rgs$

such that $(A, B) \in att$. $\mathcal{A}rgs$ is said to defend *an argument A iff every argument that attacks A is attacked by some argument in $\mathcal{A}rgs$. The* characteristic function $F : 2^{Ar} \to 2^{Ar}$ *is defined as $F(\mathcal{A}rgs) = \{A | A$ is defended by $\mathcal{A}rgs\}$. A conflict-free set $\mathcal{A}rgs$ is said to be* admissible *iff $\mathcal{A}rgs \subseteq F(\mathcal{A}rgs)$, which means that the arguments in the set can defend themselves against any attackers in the framework. Finally, we write $\mathcal{A}rgs^+ = \{A | A$ is attacked by $\mathcal{A}rgs\}$ to refer to the set of arguments attacked by $\mathcal{A}rgs$.*

The traditional approaches to argumentation semantics are based on extensions of arguments. Some of the mainstream approaches are summarized in the following definition[1].

Definition 3. *(extension-based argumentation semantics). Given an argumentation framework $AF = (Ar, att)$ and $S \subseteq Ar$:*

- *S is a complete extension of AF iff S is a conflict-free fixpoint of F (that is, if S is conflict-free and $S = F(S)$).*
- *S is a grounded extension of AF iff S is the minimal (w.r.t. set inclusion) conflict-free fixpoint of F.*
- *S is a preferred extension of AF iff S is a maximal (w.r.t. set inclusion) conflict-free fixpoint of F.*
- *S is a stable extension of AF iff S is a conflict-free fixpoint of F such that $S \cup S^+ = Ar$.*
- *S is a semi-stable extension of AF iff S is a conflict-free fixpoint of F with maximal $S \cup S^+$ (w.r.t. set inclusion).*

From Definition 3, it directly follows that every grounded, preferred, stable or semi-stable extension of a given argumentation framework is also a complete extension of that argumentation framework.

Example 1. *Let $AF = (Ar, att)$ be an abstract argumentation framework such that $Ar = \{A_1, \ldots, A_6\}$ and $att = \{(A_6, A_4), (A_4, A_6), (A_4, A_5), (A_5, A_3), (A_3, A_3)\}$. We depict AF as a directed graph, in Figure 1.*

Concerning semantics, AF has:

- Complete extensions: $\{A_1, A_2\}$, $\{A_1, A_2, A_5, A_6\}$, and $\{A_1, A_2, A_4\}$.
- Grounded extension: $\{A_1, A_2\}$.
- Preferred extensions: $\{A_1, A_2, A_5, A_6\}$, and $\{A_1, A_2, A_4\}$.
- Stable extensions: $\{A_1, A_2, A_5, A_6\}$.
- Semi-stable extensions: $\{A_1, A_2, A_5, A_6\}$.

It is worth to note that semi-stable semantics will coincide with the stable semantics whenever the framework has at least one stable extension. The reason is straightforward: A stable extension of (Ar, att) is characterized by having $S^+ = Ar \setminus S$, so $S \cup S^+ = Ar$, which implies having maximal $S \cup S^+$. As a consequence, every stable extension is also semi-stable and the existence of stable extensions is sufficient to assure no other (non-stable) semi-stable extensions exist.

[1]The characterization of the extension-based semantics in Definition 3 is slightly different than the way these were originally defined in for instance [7], but equivalence is proved in [4].

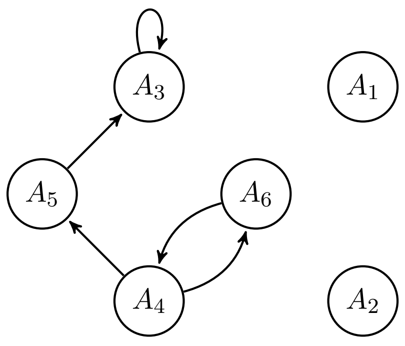


Figure 1: An abstract argumentation framework with 6 arguments.

2.2 Logic Programs and Semantics

In the current paper, we account for propositional normal logic programs, which we will call logic programs or simply programs from now on.

Definition 4. *A* rule *r is an expression* $r : c \leftarrow a_1, \ldots, a_n, \texttt{not}\ b_1, \ldots, \texttt{not}\ b_m$ *(*$n \geq 0$*,* $m \geq 0$*) where c, each* a_i *(*$1 \leq i \leq n$*) and each* b_j *(*$1 \leq j \leq m$*) are atoms and* `not` *represents negation as failure. c is called the* head *of the rule, and* $a_1, \ldots, a_n, \texttt{not}\ b_1, \ldots, \texttt{not}\ b_m$ *is called the* body *of the rule. Furthermore,* $a_1, \ldots, a_n$ *is called the* strong *part of the body and* $\texttt{not}\ b_1, \ldots, \texttt{not}\ b_m$ *is called the* weak *part of the body. Let r be a rule, we write head(r) to denote it's head (the atom c), body*$^+(r)$ *to denote the set* $\{a_1, \ldots, a_n\}$ *and body*$^-(r)$ *to denote the set* $\{\texttt{not}\ b_1, \ldots, \texttt{not}\ b_m\}$*. A logic program P is then defined as a finite set of rules. If every rule* $r \in P$ *has body*$^-(r) = \emptyset$*, we say that P is a positive program. The* Herbrand Base *of a program P is the set* HB_P *of all atoms appearing in the program.*

A wide range of logic programs semantics can be defined based on the 3-valued interpretations of programs [13] as we introduce next.

Definition 5. *A* 3-valued Herbrand interpretation *I of a logic program P is a pair* $S =< T; F >$ *of sets of atoms that suggests that the atoms in T are true and those in F are false. The atoms in* $U = HB_P \setminus (T \cup F)$ *are then suggested to be undefined in I.*

Let $I =< T; F >$ be a 3-valued interpretation of the program P, take P/I to be the program built by the execution of the following steps:

1. Remove each rule r from P that has *body*$^-(r) \cap T \neq \emptyset$;
2. From the result of step 1, remove any occurrences of atoms $\texttt{not}\ b_i$ from body of the each rule, for every $b_i \in F$.
3. From the result of step 2, substitute any occurrences of $\texttt{not}\ b_i$ left by $\mathbf{u}$.

In the above, $\mathbf{u}$ is an atom not in HB_P which is undefined in all interpretations of P (a constant). It can be observed that P/I is a positive program, since all instances of weak negation have been removed. As a consequence, P/I has a unique least 3-valued model [13] which consists of the interpretation $\Psi(I) =< T_\Psi; F_\Psi >$[2] with minimal T_Ψ and maximal F_Ψ (w.r.t. set inclusion) such that, for every $A \in HB_P$:

- $A \in T_\Psi$ if there is a rule r' in P/I with $head(r') = A$ and $body^+(r') \subseteq T_\Psi$;
- $A \in F_\Psi$ if every rule r' in P/I with $head(r') = A$ has $body^+(r') \cap F_\Psi \neq \emptyset$;
- $A \in U_\Psi$ otherwise.

With that in mind, we define the semantics of logic programs we will explore in this paper.

Definition 6. *(model-based logic programming semantics). Given a normal logic program P and a 3-valued interpretation $I =< T, F >$:*

- *I is a partial stable model of P iff $I = \Psi(I)$.*
- *I is a well-founded model of P iff I is the T part of a partial stable model of P with minimal T (w.r.t. set inclusion).*
- *I is a regular model of P iff I is the T part of a partial stable model of P with maximal T (w.r.t. set inclusion).*
- *I is a stable model of P iff I is the T part of a partial stable model of P that has $F = HB_P \setminus T$, i.e., $U = \emptyset$.*
- *I is an L-stable model of P iff I is the T part of a partial stable model of P with maximal $T \cup F$.*

Some of the definitions above are not standard in the logic programming literature, but can be found via equivalence results in different papers. The definition of a partial stable model is compatible with that in the work of Przymusinski [13] (where it is called a *3-valued stable model*). The above definition of a stable model and the well-founded model also goes back to [13]. The other definitions, namely of regular and L-stable models are based on the work of Eiter et al [8], where authors discuss partial stable models (P-stable models) and variations such as maximal (M-stable) models and least undefined (L-stable) models. In their paper [8] it is shown that for normal logic programs, P-stable models coincide with Przymusinski's original notion of 3-valued stable models [13], even though they only consider the T part of there 3-valued interpretations. The M-stable models are then defined as the maximal P-stable models and shown to be equivalent to the regular models. Our definition of L-stable models is then the same as in [8], but building on top of Przymusinski's notion of 3-valued stable models [13] instead of on to of P-stable models (notice that these are equivalent anyway).

Please notice the similarity of definitions 6 and 3. As we will show later in the paper, this similarity is not a coincidence. The following example should help the reader to further understand the above concepts.

[2]The above definition consists of a least fix-point of the immediate consequence operator Ψ defined in [13], which is guaranteed to exist and be unique for positive programs.

Example 2. *Consider the following normal logic program P with rules $\{r_1, ..., r_6\}$:*

$$\begin{aligned} r_1 :&\quad b \leftarrow c, \texttt{not}\ a \\ r_2 :&\quad a \leftarrow \texttt{not}\ b \\ r_3 :&\quad p \leftarrow c, d, \texttt{not}\ p \\ r_4 :&\quad p \leftarrow \texttt{not}\ a \\ r_5 :&\quad c \leftarrow d \\ r_6 :&\quad d \leftarrow \end{aligned}$$

The logic program from Example 2 has:

- partial stable models: $< \{d, c\}; \{\ \} >$, $< \{d, c, p, b\}; \{a\} >$, $< \{d, c, a\}; \{b\} >$.
- Well-founded model: $\{d, c\}$.
- Regular models: $\{d, c, p, b\}$, $\{d, c, a\}$.
- Stable models: $\{d, c, p, b\}$.
- L-stable models: $\{d, c, p, b\}$.

It is worth to note that L-stable semantics will coincide with the stable semantics whenever the program has one or more stable models. The reason is straightforward: Any stable model of P is a 3-valued stable model with $F = HB_P \setminus T$, so $T \cup F = HB_P$, which means maximal $T \cup F$. As a consequence, every stable model is also L-stable and the existence of stable models is sufficient to assure no other (non 2-valued stable) L-stable models exist.

Just as before, one can observe the similarity between the extensions derived in each semantics of abstract framework from Example 1 and the models from each semantics of the logic program from Example 2. Most of the similarities observed are not coincidental. In the next section we will start shedding some light on the causes of such similarities.

2.3 An Alternative Definition of AF Semantics

Given an argumentation framework AF, the function F can be computed for any conflict-free set of arguments S via a transformation of the framework: Let AF/S be the result of removing the nodes that represent arguments in S^+, we will find that $F(S)$ is exactly the set of arguments that have no attackers in AF/S (by definition). These arguments are the ones defended by S in AF. Just as well, if $F(S) = R$, then R^+ would be the set of rejected arguments in AF and the status of any others would be considered undecided (according to S).

The transformation of argumentation frameworks above is thought in the same sense as the Gelfond-Lifschitz transformation [9] that is used to define some of the most common semantics of logic programs. For sets of arguments S that are not conflict-free, this transformation may fail to provide the set of arguments defended by S. As an example, consider the framework $(\{a\}, \{(a, a)\})$, i.e., with a single self-attacking argument. The argument a attacks and defends itself, so $S = \{a\}$ is not conflict-free. The result of the transformation is an empty framework (no arguments left), since $a \in S^+$ gets removed. The conclusion is that no arguments should be accepted, not even a. However, this issue does not cause any real difficulties, since all argumentation semantics considered in the current paper satisfy the conflict-freeness property.

Theorem 7. *(alternative extension-based argumentation semantics). Given an argumentation framework $AF = (Ar, att)$, and any set of arguments S, let $G(S)$ be the set of arguments with no attackers in AF/S. Then:*

- *S is a complete extension of AF iff S is a fixpoint of G (that is, if $S = G(S)$).*
- *S is a grounded extension of AF iff S is the minimal (w.r.t. set inclusion) fixpoint of G.*
- *S is a preferred extension of AF iff S is a maximal (w.r.t. set inclusion) fixpoint of G.*
- *S is a stable extension of AF iff S is a fixpoint of G such that $S \cup S^+ = Ar$.*
- *S is a semi-stable extension of AF iff S is a fixpoint of G with maximal $S \cup S^+$ (w.r.t. set inclusion).*

Proof. We first prove that S is a complete extension of AF iff S is a fixpoint of G.
"$\Rightarrow$": Let S be a complete extension of AF. This implies that A is conflict-free. As we have observed earlier, for a conflict-free set S it holds that $F(S) = G(S)$, so from the fact that $S = F(S)$ (since S is a complete extension) it follows that $S = G(S)$, so S is a fixpoint of G.
"$\Leftarrow$": Suppose $S = G(S)$. We now prove that S is conflict-free. Suppose, towards a contradiction, that there exist $A, B \in S$ such that A attacks B (there is a conflict in S). As a consequence, $B \notin G(S)$, but since $S = G(S)$, it follows that $B \notin S$, which is a contradiction. Therefore, S is conflict-free. From our assumption that $S = G(S)$ it then follows that $S = F(S)$. Hence, S is a conflict-free fixpoint of F, that is, S is a complete extension.

The rest of the equivalences follow from the facts that S is a complete extension of AF iff S is a fixpoint of G and that grounded, preferred, stable, and semi-stable extensions are all particular cases of complete extensions. □

Please observe that Definitions 3 and Theorem 7 are related. Our theorem therefore provides an alternative definition of extension-based argumentation semantics while suggesting equivalences between semantics of logic programs and semantics of argumentation frameworks. Such equivalences will be further discussed and verified along the paper.

3 Logic Programming as Argumentation; a three-step process

In the current section, we examine how argumentation theory can be applied in the context of logic programming. In essence, our treatment is based on the approach described in [17].[3] The idea is to apply the standard three-step process of instantiated argumentation, similar to what is done in [3, 11, 10]. One starts with a particular knowledge base and constructs the associated argumentation framework (step 1), then applies abstract argumentation semantics (step 2) and subsequently looks at what the results of the argumentation semantics imply at the level of conclusions (step 3). We now specify what this process looks like in the specific context of logic programming.

[3]One particular difference is that in our current approach, arguments are recursive structures, whereas in the approach of [17], they are trees of rules. The disadvantage of the latter approach is that if one identifies the nodes of a tree with rules, one is not able to apply the same rule at different positions in the argument. The approach in the current paper, which is based on [3, 11], avoids this problem.

3.1 Step 1: Argumentation Framework Construction

The approach of instantiated argumentation starts with a particular given knowledge base. In the context of logic programming, the knowledge base consists of a logic program. For current purposes, we consider this logic program to be normal, as we defined previously. Based on a particular logic program P, one can then start to construct *arguments*, which is done in the following recursive way:

Definition 8. *Let P be a logic program.*

- *If $c \leftarrow \texttt{not}\ b_1, \ldots, \texttt{not}\ b_m$ is a rule in P then it is also an argument (say A) with*
  $\texttt{Conc}(A) = c$,
  $\texttt{Rules}(A) = \{c \leftarrow \texttt{not}\ b_1, \ldots, \texttt{not}\ b_m\}$, *and*
  $\texttt{Vul}(A) = \{b_1, \ldots, b_m\}$.

- *If $c \leftarrow a_1, \ldots, a_n, \texttt{not}\ b_1, \ldots, \texttt{not}\ b_m$ is a rule in P and for each a_i $(1 \leq i \leq n)$ there exists an argument A_i with $\texttt{Conc}(A_i) = a_i$ and $c \leftarrow a_1, \ldots, a_n, \texttt{not}\ b_1, \ldots, \texttt{not}\ b_m \not\in A_i$ then $c \leftarrow (A_1), \ldots, (A_n), \texttt{not}\ b_1, \ldots, \texttt{not}\ b_m$ is an argument (say A) with*
  $\texttt{Conc}(A) = c$,
  $\texttt{Rules}(A) = \texttt{Rules}(A_1) \cup \ldots \cup \texttt{Rules}(A_n) \cup \{c \leftarrow a_1, \ldots, a_n, \texttt{not}\ b_1, \ldots, \texttt{not}\ b_m\}$, *and*
  $\texttt{Vul}(A) = \texttt{Vul}(A_1) \cup \ldots \cup \texttt{Vul}(A_n) \cup \{b_1, \ldots, b_m\}$.

In essence, an argument can be seen as a tree-like structure of rules (the only difference with a real tree is that a rule can occur at more than one place in the argument)[4] and in examples we will often represent it as such. If A is an argument then $\texttt{Conc}(A)$ is referred to as the *conclusion* of A and $\texttt{Vul}(A)$ is referred to as the *vulnerabilities* of A.

Example 3. *Given the logic program P from 2, one can construct the following arguments:*

$$\begin{array}{ll} A_1: & d \leftarrow \\ A_2: & c \leftarrow (A_1) \\ A_3: & p \leftarrow (A_2), (A_1), \texttt{not}\ p \\ A_4: & a \leftarrow \texttt{not}\ b \\ A_5: & p \leftarrow \texttt{not}\ a \\ A_6: & b \leftarrow (A_2), \texttt{not}\ a \end{array}$$

In our example, it holds that $\texttt{Conc}(A_1) = d$, $\texttt{Conc}(A_2) = c$, $\texttt{Conc}(A_3) = \texttt{Conc}(A_5) = p$, $\texttt{Conc}(A_4) = a$, and $\texttt{Conc}(A_6) = b$. Just as well, $\texttt{Vul}(A_1) = \texttt{Vul}(A_2) = \emptyset$, $\texttt{Vul}(A_3) = \{p\}$, $\texttt{Vul}(A_4) = \{b\}$, and $\texttt{Vul}(A_5) = \texttt{Vul}(A_6) = \{a\}$. The arguments are graphically depicted in Figure 2.

We draw attention of the reader to the relations amongst these arguments. Observe, for instance, that A_1 is a subargument of A_2. Just as well, we point out that A_2 is a subargument of A_3 and A_6 and, as consequence, A_1 is a subargument of A_3 and A_6 as well.

The next step in constructing the argumentation framework is to determine the attack relation: An argument attacks another iff its conclusion is one of the vulnerabilities of the attacked argument.

[4]Notice that Definition 8 allows the same rule to occur multiple times if it is in different branches, but not if it is in the same branch. This implies that, for instance, for logic program $P = \{a \leftarrow b, c;\ b \leftarrow d;\ c \leftarrow d;\ d \leftarrow;\ e \leftarrow f;\ f \leftarrow e\ f \leftarrow\}$, $a \leftarrow (b \leftarrow (d \leftarrow)), (c \leftarrow (d \leftarrow))$ is a well-formed argument, whereas $e \leftarrow (f \leftarrow (e \leftarrow (f \leftarrow)))$ is not. This is to prevent a finite logic program from generating an infinite number of arguments, which would be particularly troublesome in the context of semi-stable semantics (see [6, 15]).

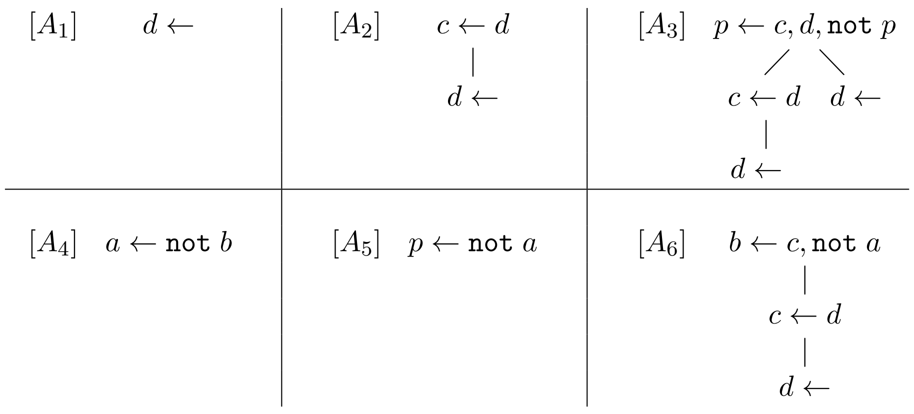

[A1] d ←
[A2] c ← d, d ←
[A3] p ← c, d, not p; c ← d; d ←; d ←
[A4] a ← not b
[A5] p ← not a
[A6] b ← c, not a; c ← d; d ←

Figure 2: Arguments constructed from P.

Definition 9. *Let A and B be arguments in the sense of Definition 8. We say that A* attacks *B iff* $\texttt{Conc}(A) \in \texttt{Vul}(B)$*.*

For the arguments of Figure 2, it holds that A_6 attacks A_4, A_4 attacks A_6 (mutual attacks), A_4 attacks A_5, A_5 attacks A_3, and A_3 attacks itself. The resulting argumentation framework (depicted in Figure 3) is essentially the same argumentation framework as in Example 1.

The notion of attack has a clear conceptual meaning. The fact that $b \in \texttt{Vul}(A)$ means that A is constructed using at least one rule containing $\texttt{not}\ b$ in its body. In essence, A is a defeasible derivation that depends on b not being derivable. An argument B that provides a (possibly defeasible) derivation of b (that is, $\texttt{Conc}(B) = b$) can therefore be seen as *attacking* A.

Using the thus defined concepts of arguments and attacks, one is then able to define the argumentation framework that is associated to a particular logic program.

Definition 10. *Let P be a logic program. We define its associated argumentation framework as $AF_P = (Ar_P, att_P)$ where Ar_P is the set of arguments in the sense of Definition 8 and att_P is the attack relation in the sense of Definition 9.*

As an example, the argumentation framework associated with the logic program of Example 2 is depicted in Figure 3.

3.2 Step 2: Applying Argumentation Semantics

Once the argumentation framework has been constructed, the next question becomes which arguments should be accepted and which arguments should be rejected. As we have seen in Section 2, several approaches have been stated in the literature for determining this. For current purposes, we will focus on the concept of complete semantics [7], which can be defined using the concept of a complete labelling [2, 4].

Definition 11. *Let $AF = (Ar, att)$ be an argumentation framework. An* argument labelling *is a function $ArgLab : Ar \to \{\texttt{in}, \texttt{out}, \texttt{undec}\}$. An argument labelling is called a* complete argument labelling *iff for each $A \in Ar$ it holds that:*

- *if $ArgLab(A) = \texttt{in}$ then for every $B \in Ar$ that attacks A it holds that $ArgLab(B) = \texttt{out}$*

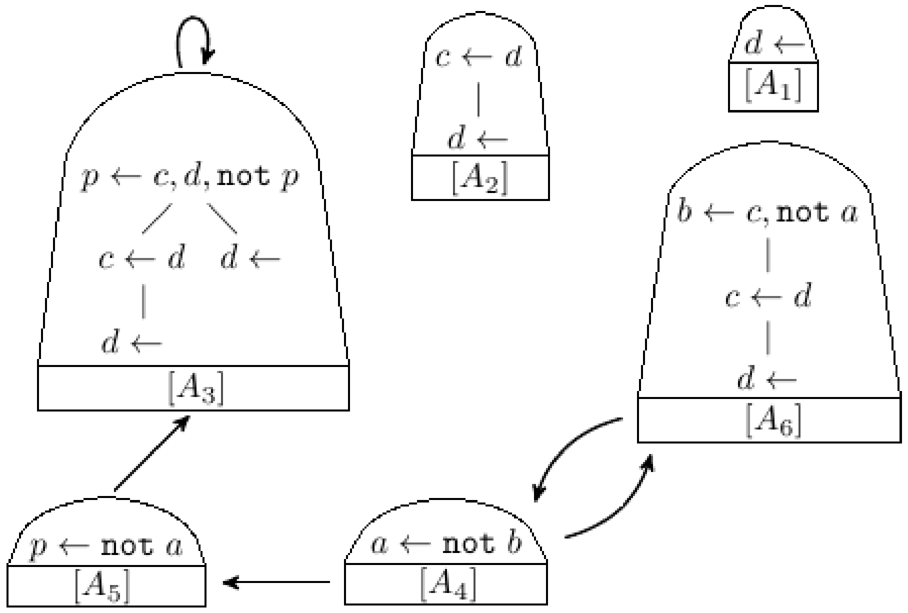


Figure 3: The abstract framework built using arguments instantiated from P

- *if* $ArgLab(A) = \texttt{out}$ *then there exists a* $B \in Ar$ *that attacks* A *such that* $ArgLab(B) = \texttt{in}$
- *if* $ArgLab(A) = \texttt{undec}$ *then (i) not every* $B \in Ar$ *that attacks* A *has* $ArgLab(B) = \texttt{out}$ *and (ii) no* $B \in Ar$ *that attacks* A *has* $ArgLab(B) = \texttt{in}$

With an argument labelling, one can express any arbitrary position on which arguments to accept (labelled in), which arguments to reject (labelled out) and which arguments to abstain from having an explicit opinion about (labelled undec). However, some of these positions are more reasonable than others. The idea of a complete labelling is that a position is reasonable iff one has sufficient reasons for each argument one accepts (all its attackers are rejected), for each argument one rejects (it has an attacker that is accepted) and for each argument one abstains (there are insufficient grounds to accept it and insufficient grounds to reject it).

When *ArgLab* is an argument labelling, we write $\texttt{in}(ArgLab)$ to denote the set of $\{A \mid ArgLab(A) = \texttt{in}\}$, $\texttt{out}(ArgLab)$ for $\{A \mid ArgLab(A) = \texttt{out}\}$ and $\texttt{undec}(ArgLab)$ for $\{A \mid ArgLab(A) = \texttt{undec}\}$. Since an argument labelling essentially defines a partition among the arguments (into a set of in-labelled arguments, a set of out-labelled arguments and a set of undec-labelled arguments), we sometimes write *ArgLab* as a triple $(\mathcal{A}rgs_1, \mathcal{A}rgs_2, \mathcal{A}rgs_3)$ where $\mathcal{A}rgs_1 = \texttt{in}(ArgLab)$, $\mathcal{A}rgs_2 = \texttt{out}(ArgLab)$ and $\mathcal{A}rgs_3 = \texttt{undec}(ArgLab)$.

In the argumentation framework of Figure 3, there are three possible complete labellings:
$ArgLab_1 = (\{A_1, A_2\}, \{\ \}, \{A_3, A_4, A_5, A_6\})$
$ArgLab_2 = (\{A_1, A_2, A_5, A_6\}, \{A_3, A_4\}, \{\ \})$
$ArgLab_3 = (\{A_1, A_2, A_4\}, \{A_5, A_6\}, \{A_3\})$

3.3 Step 3: converting argument labellings to conclusion labellings

When it comes to practical questions like what to believe or what to do, in the end what is important are not so much the arguments themselves but their conclusions. In the argumentation process, this means that for each position on which *arguments* to accept, reject or abstain we need to determine the associated position on which *conclusions* to accept, reject or abstain.

For current purposes, we follow the approach described in [16]. Here, the idea is for each conclusion to identify the "best" argument that yields it. We assume a strict total order between the different individual labels such that $\texttt{in} > \texttt{undec} > \texttt{out}$. The best argument for a particular conclusion is then the argument with the highest label. In case there is no argument at all for a particular conclusion, the conclusion will then simply be labelled `out`.

Definition 12 ([16]). *Let P be a logic program. A conclusion labelling is a function $\mathit{ConcLab} : \mathit{HB}_P \rightarrow \{\texttt{in}, \texttt{out}, \texttt{undec}\}$ where HB_P is the set of all atoms occurring in P.*
Let $\mathit{AF}_P = (\mathit{Ar}_P, \mathit{att}_P)$ be an argumentation framework and ArgLab be an argument labelling of AF_P. We say that $\mathit{ConcLab}$ is the associated conclusion labelling *of ArgLab iff $\mathit{ConcLab}$ is a conclusion labelling such that for each $c \in \mathit{HB}_P$ it holds that $\mathit{ConcLab}(c) = \mathit{max}(\{\mathit{ArgLab}(A) \mid \texttt{Conc}(A) = c\} \cup \{\texttt{out}\})$ where $\texttt{in} > \texttt{undec} > \texttt{out}$. We say that a conclusion labelling is* complete *iff it is associated with a complete argument labelling.*

When $\mathit{ConcLab}$ is a conclusion labelling, we write $\texttt{in}(\mathit{ConcLab})$ to denote the set $\{c \mid \mathit{ConcLab}(c) = \texttt{in}\}$, $\texttt{out}(\mathit{ConcLab})$ for $\{c \mid \mathit{ConcLab}(c) = \texttt{out}\}$ and $\texttt{undec}(\mathit{ConcLab})$ for $\{c \mid \mathit{ConcLab}(c) = \texttt{undec}\}$. Since a conclusion labelling essentially defines a partition among HB_P (into a set of `in`-labelled conclusions, a set of `out`-labelled conclusions and a set of `undec`-labelled conclusions), we sometimes write $\mathit{ConcLab}$ as a triple $(\mathcal{C}\mathit{oncs}_1, \mathcal{C}\mathit{oncs}_2, \mathcal{C}\mathit{oncs}_3)$ where $\mathcal{C}\mathit{oncs}_1 = \texttt{in}(\mathit{ConcLab})$, $\mathcal{C}\mathit{oncs}_2 = \texttt{out}(\mathit{ConcLab})$ and $\mathcal{C}\mathit{oncs}_3 = \texttt{undec}(\mathit{ConcLab})$.

Recall that for the argumentation framework in Figure 3 the complete argument labellings are: $\mathit{ArgLab}_1 = (\{A_1, A_2\}, \{\ \}, \{A_3, A_4, A_5, A_6\})$, $\mathit{ArgLab}_2 = (\{A_1, A_2, A_5, A_6\}, \{A_3, A_4\}, \{\ \})$, and $\mathit{ArgLab}_3 = (\{A_1, A_2, A_4\}, \{A_5, A_6\}, \{A_3\})$. In the same example, observe that two of the involved arguments yield conclusion p, namely $A_3 :\ p \leftarrow (A_5), (A_6), \texttt{not}\ p$ and $A_5 : p \leftarrow \texttt{not}\ a$. Observe that A_3 is labelled `out`, while A_5 is labelled `in` by ArgLab_2. As a consequence, the label of p in the conclusion labelling associated with ArgLab_2 is `in`. Using a similar reasoning, we obtain that the associated conclusion labelling $\mathit{ConcLab}_1$ of ArgLab_1 is $(\{c, d\}, \{\ \}, \{p, a, b\})$, the associated conclusion labelling $\mathit{ConcLab}_2$ of argument labelling ArgLab_2 is $(\{c, d, p, b\}, \{a\}, \{\ \})$, and the associated conclusion labelling $\mathit{ConcLab}_2$ of argument labelling ArgLab_2 is $(\{c, d, a\}, \{b\}, \{p\})$.

4 On the Minimization and Maximization of Argument Labellings

Now that the general overview of the three-step process has been provided, we will subsequently zoom in on some of its steps, starting with the argument labellings level (step 2). In particular, we provide a brief overview (based on work originally published in [2, 4]) of how the different argument labellings relate to each other, especially when it comes to maximizing or minimizing a particular label.

However, we start with an alternative way to characterize the concept of a complete argument labelling, which will be applied several times in subsequent proofs.

Proposition 1. *Let $\mathit{AF} = (\mathit{Ar}, \mathit{att})$ be an argumentation framework. An argument labelling ArgLab is a complete argument labelling iff for each $A \in \mathit{Ar}$ it holds that:*

- *if for every $B \in Ar$ that attacks A it holds that $ArgLab(B) = \texttt{out}$, then $ArgLab(A) = \texttt{in}$*
- *if there exists a $B \in Ar$ that attacks A such that $ArgLab(B) = \texttt{in}$, then $ArgLab(A) = \texttt{out}$*
- *if not for every $B \in Ar$ that attacks A it holds that $ArgLab(B) = \texttt{out}$ and there does not exist a $B \in Ar$ that attacks A such that $ArgLab(B) = \texttt{in}$, then $ArgLab(A) = \texttt{undec}$*

Lemma 1 ([2, 4])**.** *Let $ArgLab_1$ and $ArgLab_2$ be complete argument labellings of argumentation framework $AF = (Ar, att)$. It holds that*

- $\texttt{in}(ArgLab_1) \subseteq \texttt{in}(ArgLab_2)$ *iff* $\texttt{out}(ArgLab_1) \subseteq \texttt{out}(ArgLab_2)$
- $\texttt{in}(ArgLab_1) \subsetneq \texttt{in}(ArgLab_2)$ *iff* $\texttt{out}(ArgLab_1) \subsetneq \texttt{out}(ArgLab_2)$
- $\texttt{in}(ArgLab_1) = \texttt{in}(ArgLab_2)$ *iff* $\texttt{out}(ArgLab_1) = \texttt{out}(ArgLab_2)$

From Lemma 1 it immediately follows that $\texttt{in}(ArgLab_1) = \texttt{in}(ArgLab_2)$ iff $ArgLab_1 = ArgLab_2$, and that $\texttt{out}(ArgLab_1) = \texttt{out}(ArgLab_2)$ iff $ArgLab_1 = ArgLab_2$.

From Lemma 1 one can then obtain the following result.

Theorem 13 ([2, 4])**.** *Let $ArgLab$ be a complete argument labelling of argumentation framework $AF = (Ar, att)$. It holds that*

- $\texttt{in}(ArgLab)$ *is maximal (w.r.t. set-inclusion) among all complete argument labellings of AF iff* $\texttt{out}(ArgLab)$ *is maximal (w.r.t. set-inclusion) among all complete argument labellings of AF.*
- $\texttt{in}(ArgLab)$ *is minimal (w.r.t. set-inclusion) among all complete argument labellings of AF iff* $\texttt{out}(ArgLab)$ *is minimal (w.r.t. set-inclusion) among all complete argument labellings of AF.*

Lemma 2 ([2, 4])**.** *Let $ArgLab_1$ and $ArgLab_2$ be complete argument labellings of argumentation framework $AF = (Ar, att)$. It holds that*

- *if* $\texttt{in}(ArgLab_1) \subseteq \texttt{in}(ArgLab_2)$ *then* $\texttt{undec}(ArgLab_1) \supseteq \texttt{undec}(ArgLab_2)$
- *if* $\texttt{out}(ArgLab_1) \subseteq \texttt{out}(ArgLab_2)$ *then* $\texttt{undec}(ArgLab_1) \supseteq \texttt{undec}(ArgLab_2)$
- *if* $\texttt{in}(ArgLab_1) \subsetneq \texttt{in}(ArgLab_2)$ *then* $\texttt{undec}(ArgLab_1) \supsetneq \texttt{undec}(ArgLab_2)$
- *if* $\texttt{out}(ArgLab_1) \subsetneq \texttt{out}(ArgLab_2)$ *then* $\texttt{undec}(ArgLab_1) \supsetneq \texttt{undec}(ArgLab_2)$

From Lemma 2, the following result follows.

Theorem 14. *Let $ArgLab$ be a complete argument labelling of argumentation framework $AF = (Ar, att)$. It holds that*

1. *if* $\texttt{undec}(ArgLab)$ *is minimal (w.r.t. set-inclusion) among all complete argument labellings of AF then* $\texttt{in}(ArgLab)$ *and* $\texttt{out}(ArgLab)$ *are maximal (w.r.t. set-inclusion) among all complete argument labellings of AF, and*
2. *if* $\texttt{undec}(ArgLab)$ *is maximal (w.r.t. set-inclusion) among all complete argument labellings of AF then* $\texttt{in}(ArgLab)$ *and* $\texttt{out}(ArgLab)$ *are minimal (w.r.t. set-inclusion) among all complete argument labellings of AF.*

Proof. 1. Let $ArgLab$ be a complete argument labelling where $\texttt{undec}(ArgLab)$ is minimal. That is, for each complete argument labelling $ArgLab'$ of AF, it holds that if $\texttt{undec}(ArgLab') \subseteq \texttt{undec}(ArgLab)$ then $\texttt{undec}(ArgLab) \subseteq \texttt{undec}(ArgLab')$. In order to show that $\texttt{in}(ArgLab)$ is maximal, we need to prove that for each complete argument labelling $ArgLab'$, if $\texttt{in}(ArgLab) \subseteq \texttt{in}(ArgLab')$ then $\texttt{in}(ArgLab') \subseteq \texttt{in}(ArgLab)$. Let $ArgLab'$ be a complete argument labelling such that $\texttt{in}(ArgLab) \subseteq \texttt{in}(ArgLab')$. From Lemma 2 it then follows that $\texttt{undec}(ArgLab) \supseteq \texttt{undec}(ArgLab')$. This, together with our initial assumption that $\texttt{undec}(ArgLab)$ is minimal, implies that $\texttt{undec}(ArgLab) = \texttt{undec}(ArgLab')$. Therefore, it cannot be the case that $\texttt{in}(ArgLab) \subsetneq \texttt{in}(ArgLab')$ (because otherwise $\texttt{undec}(ArgLab) \supsetneq \texttt{undec}(ArgLab')$ would follow from Lemma 2, which would conflict with the fact that $\texttt{undec}(ArgLab) = \texttt{undec}(ArgLab')$). This, together with the fact that $\texttt{in}(ArgLab) \subseteq \texttt{in}(ArgLab')$ implies that $\texttt{in}(ArgLab) = \texttt{in}(ArgLab')$, so $\texttt{in}(ArgLab') \subseteq \texttt{in}(ArgLab)$. The case of maximality of $\texttt{out}(ArgLab)$ can be proved in a similar way.

2. Similar to the first point.

□

Theorem 15. *Let $AF = (Ar, att)$ be an argumentation framework. The complete argument labelling $ArgLab$ where $\texttt{in}(ArgLab)$ is minimal (w.r.t. set inclusion) among all complete argument labellings is unique.*

Proof. Let $ArgLab_1$ and $ArgLab_2$ be two complete argument labellings where both $\texttt{in}(ArgLab_1)$ and $\texttt{in}(ArgLab_2)$ are minimal. We will now prove that $ArgLab_1 = ArgLab_2$. First we observe that from Theorem 13, it follows that also $\texttt{out}(ArgLab_1)$ and $\texttt{out}(ArgLab_2)$ are minimal. Let us now define $ArgLab$ to be the outcome of the skeptical judgment aggregation operator of [5, Def. 18]. It holds that $ArgLab$ is a complete argument labelling [5, Th. 8] such that $\texttt{in}(ArgLab) \subseteq \texttt{in}(ArgLab_1)$, $\texttt{out}(ArgLab) \subseteq \texttt{out}(ArgLab_1)$, $\texttt{in}(ArgLab) \subseteq \texttt{in}(ArgLab_2)$ and $\texttt{out}(ArgLab) \subseteq \texttt{out}(ArgLab_2)$ ([5, Th. 7]). The fact that $ArgLab_1$ and $ArgLab_2$ are complete argument labellings with minimal `in` and minimal `out` then implies that $\texttt{in}(ArgLab) = \texttt{in}(ArgLab_1)$, $\texttt{out}(ArgLab) = \texttt{out}(ArgLab_1)$, $\texttt{in}(ArgLab) = \texttt{in}(ArgLab_2)$ and $\texttt{out}(ArgLab) = \texttt{out}(ArgLab_2)$, so $ArgLab = ArgLab_1$ and $ArgLab = ArgLab_2$, so $ArgLab_1 = ArgLab_2$. □

Theorem 16. *Let $AF = (Ar, att)$ be an argumentation framework and $ArgLab$ be one of its complete argument labellings. It holds that $\texttt{undec}(ArgLab)$ is maximal (w.r.t. set-inclusion) among all complete argument labellings iff $\texttt{in}(ArgLab)$ is minimal (w.r.t. set-inclusion) among all complete argument labellings.*

Proof. "⇒": This follows from Theorem 14.
"⇐": Let $ArgLab$ be the unique (Theorem 15) complete argument labelling where $\texttt{in}(ArgLab)$ is minimal. That is, for any complete argument labelling $ArgLab'$ it holds that $\texttt{in}(ArgLab) \subseteq \texttt{in}(ArgLab')$, so we also have (by Lemma 2) $\texttt{undec}(ArgLab) \supseteq \texttt{undec}(ArgLab')$. Therefore, $ArgLab$ is also a complete argument labelling where $\texttt{undec}(ArgLab)$ is maximal. □

To summarize, the complete argument labellings where `in` is maximal are the same as the complete argument labellings where `out` is maximal (Theorem 13). These argument labellings will be referred to as *preferred argument labellings*. Furthermore, the unique complete argument labelling where `in` is minimal (Theorem 15) is the same as the unique complete argument labelling where `out` is minimal (Theorem 13) and the same as the unique complete argument labelling where undec is maximal (Theorem 16). This argument labelling will be referred to as the *grounded argument labelling.* The complete argument labellings where undec is minimal will be referred to as *semi-stable argument labellings.* The complete argument labellings where undec is empty will be referred to as *argstable argument labellings.* In fact, from our results, it follows that (*i*) every argstable argument labelling is also semi-stable and (*ii*) every semi-stable argument labelling is also preferred. These kinds of argument labellings are further summarized in Table 1.

Condition	Resulting Argument Labelling
NONE	Complete
MAX In	Preferred
MAX Out	Preferred
MAX Undec	Grounded
MIN In	Grounded
MIN Out	Grounded
MIN Undec	Semi-stable
NO Undec	Argstable

Table 1: Kinds of argument labellings

It is relatively straightforward to define associated classes of conclusion labellings. For instance, a conclusion labelling is said to be a *preferred conclusion labelling* iff it is the associated conclusion labelling of a preferred argument labelling. Similarly, a conclusion labelling is said to be the *grounded conclusion labelling* iff it is the associated conclusion labelling of the grounded argument labelling, and a conclusion labelling is said to be a *semi-stable conclusion labelling* iff it is the associated conclusion labelling of a semi-stable argument labelling. Just as well, a conclusion labelling is said to be an *argstable conclusion labelling* iff it is the associated conclusion labelling of an argstable argument labelling.

5 On the Minimization and Maximization of Conclusion Labellings

The concepts of preferred, grounded, semi-stable, and argstable conclusion labellings, as defined at the end of the previous section, are based on the common idea of performing the maximization/minimization at the level of argument labellings and then identifying the associated conclusion labellings. An alternative procedure would be simply to identify *all* complete conclusion labellings and then to perform the maximization/minimization right at the level of the conclusion labellings. In the current section, we analyze this alternative procedure. We will observe that the thus derived conclusion labellings relate to each other in a way that is very similar to how the different types of argument labellings of the previous section relate to each other.

Lemma 3. *Let $ConcLab_1$ and $ConcLab_2$ be complete conclusion labellings of logic program P and associated argumentation framework $AF_P = (Ar_P, att_P)$. It holds that*

1. $\texttt{in}(ConcLab_1) \subseteq \texttt{in}(ConcLab_2)$ *iff* $\texttt{out}(ConcLab_1) \subseteq \texttt{out}(ConcLab_2)$
2. $\texttt{in}(ConcLab_1) = \texttt{in}(ConcLab_2)$ *iff* $\texttt{out}(ConcLab_1) = \texttt{out}(ConcLab_2)$

3. $\texttt{in}(ConcLab_1) \subsetneq \texttt{in}(ConcLab_2)$ *iff* $\texttt{out}(ConcLab_1) \subsetneq \texttt{out}(ConcLab_2)$

Proof. Let $ArgLab_1$ be a complete argument labelling of which $ConcLab_1$ is the associated conclusion labelling, and let $ArgLab_2$ be a complete argument labelling of which $ConcLab_2$ is the associated conclusion labelling.

1. "$\Rightarrow$": Suppose $\texttt{in}(ConcLab_1) \subseteq \texttt{in}(ConcLab_2)$. Let $c \in \texttt{out}(ConcLab_1)$. We will now prove that $c \in \texttt{out}(ConcLab_2)$. First, we observe that the fact that $c \in \texttt{out}(ConcLab_1)$ implies (Definition 12) that for each argument A such that $\texttt{Conc}(A) = c$ it holds that $ArgLab_1(A) = \texttt{out}$. This implies (Definition 11) that each such A has an attacker (say B) such that $ArgLab_1(B) = \texttt{in}$, which implies that (Definition 12) that $ConcLab_1(\texttt{Conc}(B)) = \texttt{in}$. Given the assumption that $\texttt{in}(ConcLab_1) \subseteq \texttt{in}(ConcLab_2)$, it then follows that $ConcLab_2(\texttt{Conc}(B)) = \texttt{in}$, which (Definition 12) implies that there exists an argument $C \in Ar_P$ with $\texttt{Conc}(C) = \texttt{Conc}(B)$ and $ArgLab_2(C) = \texttt{in}$. Since the notion of attack is based on the conclusion of the attacking argument (Definition 9), it follows that C attacks A. Since C is labelled $\texttt{in}$ by $ArgLab_2$, it follows (Definition 11) that A is labelled $\texttt{out}$ by $ArgLab_2$. Since this holds for any argument A with conclusion c, it follows (Definition 12) that $ConcLab_2(c) = \texttt{out}$. That is, $c \in \texttt{out}(ConcLab_2)$.
   "$\Leftarrow$": Suppose $\texttt{out}(ConcLab_1) \subseteq \texttt{out}(ConcLab_2)$. Let $c \in \texttt{in}(ConcLab_1)$. First, we observe that the fact that $c \in \texttt{in}(ConcLab_1)$ implies (Definition 12) that there is an argument (say A) such that $\texttt{Conc}(A) = c$ and $ArgLab_1(A) = \texttt{in}$. This implies (Definition 11) that for each attacker B of A it holds that $ArgLab_1(B) = \texttt{out}$. This means that (Definition 9) for each argument B with $\texttt{Conc}(B) \in \texttt{Vul}(A)$ it holds that $ArgLab_1(B) = \texttt{out}$, which then implies (Definition 12) that for each $b \in \texttt{Vul}(A)$ it holds that $ConcLab_1(b) = \texttt{out}$. From our initial assumption that $\texttt{out}(ConcLab_1) \subseteq \texttt{out}(ConcLab_2)$, it then follows that $ConcLab_2(b) = \texttt{out}$. So for every $B \in Ar_P$ with $\texttt{Conc}(B) = b$, it holds that $ArgLab_2(B) = \texttt{out}$ (Definition 12), which implies that all attackers of A are labelled $\texttt{out}$ by $ArgLab_2$ (Definition 9). Hence, it follows that A is labelled $\texttt{in}$ by $ArgLab_2$ (Definition 1). Since $\texttt{Conc}(A) = c$, it follows (Definition 12) that $ConcLab_2(c) = \texttt{in}$. That is, $c \in \texttt{in}(ConcLab_2)$.

2. This follows from point 1.

3. This follows from point 1 and point 2.

□

From Lemma 3 it immediately follows that $\texttt{in}(ConcLab_1) = \texttt{in}(ConcLab_2)$ iff $ConcLab_1 = ConcLab_2$, and, as well, that $\texttt{out}(ConcLab_1) = \texttt{out}(ConcLab_2)$ iff $ConcLab_1 = ConcLab_2$.

One can then obtain the following result:

Theorem 17. *Let ConcLab be a complete conclusion labelling of logic program P and the associated argumentation framework $AF_P = (Ar_P, att_P)$. It holds that*

- $\texttt{in}(ConcLab)$ *is maximal (w.r.t. set-inclusion) among all complete conclusion labellings of AF iff* $\texttt{out}(ConcLab)$ *is maximal (w.r.t. set-inclusion) among all complete conclusion labellings of AF.*

- $\texttt{in}(ConcLab)$ *is minimal (w.r.t. set-inclusion) among all complete conclusion labellings of AF iff* $\texttt{out}(ConcLab)$ *is minimal (w.r.t. set-inclusion) among all complete conclusion labellings of AF.*

Proof. This follows directly from Lemma 3. □

Lemma 4. *Let $ConcLab_1$ and $ConcLab_2$ be complete conclusion labellings of logic program P and the associated argumentation framework $AF_P = (Ar_P, att_P)$. It holds that*

- *if* $\texttt{in}(ConcLab_1) \subseteq \texttt{in}(ConcLab_2)$ *then* $\texttt{undec}(ConcLab_1) \supseteq \texttt{undec}(ConcLab_2)$
- *if* $\texttt{out}(ConcLab_1) \subseteq \texttt{out}(ConcLab_2)$ *then* $\texttt{undec}(ConcLab_1) \supseteq \texttt{undec}(ConcLab_2)$
- *if* $\texttt{in}(ConcLab_1) \subsetneq \texttt{in}(ConcLab_2)$ *then* $\texttt{undec}(ConcLab_1) \supsetneq \texttt{undec}(ConcLab_2)$
- *if* $\texttt{out}(ConcLab_1) \subsetneq \texttt{out}(ConcLab_2)$ *then* $\texttt{undec}(ConcLab_1) \supsetneq \texttt{undec}(ConcLab_2)$

Proof.

1. Suppose that $\texttt{in}(ConcLab_1) \subseteq \texttt{in}(ConcLab_2)$. Then (Lemma 3) it follows that $\texttt{out}(ConcLab_1) \subseteq \texttt{out}(ConcLab_2)$. Considering that $(\texttt{in}(ConcLab_1), \texttt{out}(ConcLab_1), \texttt{undec}(ConcLab_1))$ and $(\texttt{in}(ConcLab_2), \texttt{out}(ConcLab_2), \texttt{undec}(ConcLab_2))$ are partitions, and given that $\texttt{in}(ConcLab_1) \subseteq \texttt{in}(ConcLab_2)$ and $\texttt{out}(ConcLab_1) \subseteq \texttt{out}(ConcLab_2)$, we conclude that $\texttt{undec}(ConcLab_1) \supseteq \texttt{undec}(ConcLab_2)$.
2. Similar to the first point.
3. Suppose that $\texttt{in}(ConcLab_1) \subsetneq \texttt{in}(ConcLab_2)$. Then (by Lemma 3) it follows that $\texttt{out}(ConcLab_1) \subsetneq \texttt{out}(ConcLab_2)$. Considering that $(\texttt{in}(ConcLab_1), \texttt{out}(ConcLab_1), \texttt{undec}(ConcLab_1))$ and $(\texttt{in}(ConcLab_2), \texttt{out}(ConcLab_2), \texttt{undec}(ConcLab_2))$ are partitions, together with the facts that $\texttt{in}(ConcLab_1) \subsetneq \texttt{in}(ConcLab_2)$ and $\texttt{out}(ConcLab_1) \subsetneq \texttt{out}(ConcLab_2)$, implies that $\texttt{undec}(ConcLab_1) \supsetneq \texttt{undec}(ConcLab_2)$.
4. Similar to the third point.

□

From Lemma 4, the following result follows.

Theorem 18. *Let $ConcLab$ be a complete conclusion labelling of logic program P and associated argumentation framework $AF_P = (Ar_P, att_P)$. It holds that*

1. *if* $\texttt{undec}(ConcLab)$ *is minimal (w.r.t. set-inclusion) among all complete conclusion labellings of AF then* $\texttt{in}(ConcLab)$ *and* $\texttt{out}(ConcLab)$ *are maximal (w.r.t. set-inclusion) among all complete conclusion labellings of AF, and*
2. *if* $\texttt{undec}(ConcLab)$ *is maximal (w.r.t. set-inclusion) among all complete conclusion labellings of AF then* $\texttt{in}(ConcLab)$ *and* $\texttt{out}(ConcLab)$ *are minimal (w.r.t. set-inclusion) among all complete conclusion labellings of AF.*

Proof.

1. Suppose $ConcLab$ is a complete conclusion labelling such that $\texttt{undec}(ConcLab)$ is minimal. As a consequence, for each complete conclusion labelling $ConcLab'$, if $\texttt{undec}(ConcLab') \subseteq \texttt{undec}(ConcLab)$ then $\texttt{undec}(ConcLab) \subseteq \texttt{undec}(ConcLab')$. In order to prove that $\texttt{in}(ConcLab)$ is maximal, we need to prove that for each complete conclusion labelling $ConcLab'$, if $\texttt{in}(ConcLab) \subseteq \texttt{in}(ConcLab')$ then $\texttt{in}(ConcLab') \subseteq \texttt{in}(ConcLab)$. Suppose that $\texttt{in}(ConcLab) \subseteq \texttt{in}(ConcLab')$ for some complete conclusion labelling $ConcLab'$. It follows (Lemma 4) that $\texttt{undec}(ConcLab) \supseteq \texttt{undec}(ConcLab')$. From our initial assumption, one can conclude that $\texttt{undec}(ConcLab) \subseteq \texttt{undec}(ConcLab')$,

so we have $\mathtt{undec}(\mathit{ConcLab}) = \mathtt{undec}(\mathit{ConcLab}')$ as consequence. This means it cannot be the case that $\mathtt{in}(\mathit{ConcLab}) \subsetneq \mathtt{in}(\mathit{ConcLab}')$ (since otherwise Lemma 4 would imply that $\mathtt{undec}(\mathit{ConcLab}) \supsetneq \mathtt{undec}(\mathit{ConcLab}')$) so $\mathtt{in}(\mathit{ConcLab}) = \mathtt{in}(\mathit{ConcLab}')$, so $\mathtt{in}(\mathit{ConcLab}') \subseteq \mathtt{in}(\mathit{ConcLab})$. From the thus obtained fact that *ConcLab* has maximal `in`, it follows (Theorem 17) that *ConcLab* also has maximal `out`.

2. Similar to the first point.

□

Theorem 19. *Let P be a logic program and $AF_P = (Ar_P, att_P)$ be its associated argumentation framework. The complete conclusion labelling ConcLab of AF_P where* $\mathtt{in}(\mathit{ConcLab})$ *is minimal (w.r.t. set inclusion) among all complete conclusion labellings of AF_P is unique.*

Proof. It suffices to show that the associated conclusion labelling of the grounded argument labelling has a set of `in`-labelled conclusions that is a subset of the set of `in`-labelled conclusions of any arbitrary complete conclusion labelling. Let ArgLab_{gr} be the grounded argument labelling of AF_P and $\mathit{ConcLab}_{gr}$ be its associated conclusion labelling (that is, $\mathit{ConcLab}_{gr}$ is the grounded conclusion labelling). We need to prove that for any complete conclusion labelling *ConcLab* it holds that $\mathtt{in}(\mathit{ConcLab}_{gr}) \subseteq \mathtt{in}(\mathit{ConcLab})$. Let *ConcLab* be an arbitrary complete conclusion labelling of P and AF_P. Assume it is the associated conclusion labelling of complete argument labelling *ArgLab*. From the fact that ArgLab_{gr} is the grounded argument labelling, it follows (Theorem 15) that $\mathtt{in}(\mathit{ArgLab}_{gr}) \subseteq \mathtt{in}(\mathit{ArgLab})$. Let $c \in \mathtt{in}(\mathit{ConcLab}_{gr})$. Then (Definition 12) there exists an argument $A \in Ar_P$ with $\mathtt{Conc}(A) = c$ and $\mathit{ArgLab}_{gr}(A) = \mathtt{in}$. From the fact that $\mathtt{in}(\mathit{ArgLab}_{gr}) \subseteq \mathtt{in}(\mathit{ArgLab})$ it follows that $\mathit{ArgLab}(A) = \mathtt{in}$, so (Definition 12) $\mathit{ConcLab}(c) = \mathtt{in}$. That is, $c \in \mathtt{in}(\mathit{ConcLab})$. So $\mathtt{in}(\mathit{ConcLab}_{gr}) \subseteq \mathtt{in}(\mathit{ConcLab})$. □

Theorem 20. *Let P be a logic program, $AF_P = (Ar_P, att_P)$ be the associated argumentation framework of P and ConcLab be one of the complete conclusion labellings of AF_P. It holds that* $\mathtt{undec}(\mathit{ConcLab})$ *is maximal (w.r.t. set-inclusion) among all complete conclusion labellings of AF_P iff* $\mathtt{in}(\mathit{ConcLab})$ *is minimal (w.r.t. set-inclusion) among all complete conclusion labellings of AF_P.*

Proof. "⇒": This follows from Theorem 18.
"⇐": Let *ConcLab* be the unique (Theorem 19) complete conclusion labelling with minimal $\mathtt{in}(\mathit{ConcLab})$. That is, for any complete conclusion labelling $\mathit{ConcLab}'$ it holds that $\mathtt{in}(\mathit{ConcLab}) \subseteq \mathtt{in}(\mathit{ConcLab}')$, so (by Lemma 4) $\mathtt{undec}(\mathit{ConcLab}) \supseteq \mathtt{undec}(\mathit{ConcLab}')$. Therefore, *ConcLab* is also a complete conclusion labelling where $\mathtt{undec}(\mathit{ConcLab})$ is maximal. □

To summarize, the complete conclusion labellings where `in` is maximal are the same as the complete conclusion labellings where `out` is maximal (Theorem 17). These argument labellings will be referred to as *regular conclusion labellings.* Furthermore, the unique complete conclusion labelling where `in` is minimal (Theorem 19) is the same as the unique complete conclusion labelling where `out` is minimal (Theorem 17) and the same as the unique complete conclusion labelling where `undec` is maximal (Theorem 20). This conclusion labelling will be referred to as the *well-founded conclusion labelling.* The complete conclusion labellings where `undec` is minimal will be referred to as *L-stable conclusion labellings.* Just as well, the complete conclusion labellings where `undec` is empty will be referred to as *concstable conclusion labellings.* In fact, from our results, it follows that (i) every concstable conclusion labelling is also L-stable and (*ii*) every L-stable conclusion labelling is also a regular conclusion labelling. These kinds of conclusion labellings are further summarized in Table 2.

Condition	Resulting Conclusion Labelling
NONE	Complete
MAX In	Regular
MAX Out	Regular
MAX Undec	Well-founded
MIN In	Well-founded
MIN Out	Well-founded
MIN Undec	L-stable
NO Undec	Concstable

Table 2: Kinds of conclusion labellings derived from programs

6 Maximizing/Minimizing Argument Labellings vs. Maximizing/Minimizing Conclusion Labellings

So far, we have described two ways of selecting particular subsets of the complete conclusion labellings:

1. Perform minimization (resp. maximization) of a particular label at the level of complete argument labellings, then determine the associated conclusion labellings. This is the approach sketched in Section 4.
2. Take all complete conclusion labellings (these are the associated labellings of *all* complete argument labellings) and then perform the minimization (resp. maximization) of a particular label at the level of complete conclusion labellings. This is the approach sketched in Section 5.

An interesting question is whether the outcome of the two procedures is actually the same. That is, does minimizing (resp. maximizing) a particular label at the level of complete argument-labellings yield the same result as minimizing (resp. maximizing) the label at the level of complete conclusion labellings? We will see that, in general, the answer is "yes", with one notable exception.

We first formally define two functions between argument labellings and conclusion labellings.

Definition 21. *Let P be a logic program and AF_P be its associated argumentation framework. Let ArgLabs be the set of all argument labellings of AF_P and let ConcLabs be the set of all conclusion labellings of P and AF_P.*

We define a function $\texttt{ArgLab2ConcLab} : \mathit{ArgLabs} \to \mathit{ConcLabs}$ *such that for each* $\mathit{ArgLab} \in \mathit{ArgLabs}$*, it holds that* $\texttt{ArgLab2ConcLab}(\mathit{ArgLab})$ *is the associated conclusion labelling of* ArgLab.

We define a function $\texttt{ConcLab2ArgLab} : \mathit{ConcLabs} \to \mathit{ArgLabs}$ *such that for each* $\mathit{ConcLab} \in \mathit{ConcLabs}$ *and each* $A \in Ar_P$ *it holds that:*
$\texttt{ConcLab2ArgLab}(\mathit{ConcLab})(A) = \texttt{in}$ *iff for each* $v \in \texttt{Vul}(A)$ *it holds that* $\mathit{ConcLab}(v) = \texttt{out}$

$\texttt{ConcLab2ArgLab}(\mathit{ConcLab})(A) = \texttt{out}$ *iff there exists a* $v \in \texttt{Vul}(A)$ *such that* $\mathit{ConcLab}(v) = \texttt{in}$
$\texttt{ConcLab2ArgLab}(\mathit{ConcLab})(A) = \texttt{undec}$ *iff not for each* $v \in \texttt{Vul}(A)$ *it holds that* $\mathit{ConcLab}(v) = \texttt{out}$ *and there does not exist a* $v \in \texttt{Vul}(A)$ *such that* $\mathit{ConcLab}(v) = \texttt{in}$

Theorem 22. *When restricted to complete argument labellings and complete conclusion labellings, the functions* `ArgLab2ConcLab` *and* `ConcLab2ArgLab` *are bijections and each other's inverse.*

Proof. It suffices to show that for each complete argument labelling *ArgLab*, it holds that $\texttt{ConcLab2ArgLab}(\texttt{ArgLab2ConcLab}(\mathit{ArgLab})) = \mathit{ArgLab}$. In that sense, consider *ConcLab* is $\texttt{ArgLab2ConcLab}(\mathit{ArgLab})$, and let ArgLab' be $\texttt{ConcLab2ArgLab}(\mathit{ConcLab})$. Our aim is to show that $\mathit{ArgLab}' = \mathit{ArgLab}$.

We first prove that if $\mathit{ArgLab}'(A) = \texttt{in}$ then $\mathit{ArgLab}(A) = \texttt{in}$. Let A be an argument such that $\mathit{ArgLab}'(A) = \texttt{in}$. Then, by definition of `ConcLab2ArgLab`, for each $v \in \texttt{Vul}(A)$ it holds that $\mathit{ConcLab}(v) = \texttt{out}$. It then follows (by definition of `ArgLab2ConcLab`) that each argument with conclusion v is labelled `out` by *ArgLab*. This (Definition 9) means that all attackers of A are labelled `out` by *ArgLab*, which then implies (Lemma 1) that A is labelled `in` by *ArgLab*. That is, $\mathit{ArgLab}(A) = \texttt{in}$.

The next thing to prove is that if $\mathit{ArgLab}'(A) = \texttt{out}$ then $\mathit{ArgLab}(A) = \texttt{out}$. Let A be an argument such that $\mathit{ArgLab}'(A) = \texttt{out}$. Then, by definition of `ConcLab2ArgLab`, there exists a $v \in \texttt{Vul}(A)$ such that $\mathit{ConcLab}(v) = \texttt{in}$. It then follows (by definition of `ArgLab2ConcLab`) that there exists an argument (say B) with $\texttt{Conc}(B) = v$ and $\mathit{ArgLab}(B) = \texttt{in}$. This then implies (Definition 9) that A has an attacker (B) that is labelled `in` by *ArgLab*, which then implies (Lemma 1) that A is labelled `out` by *ArgLab*. That is, $\mathit{ArgLab}(A) = \texttt{out}$.

Finally, we prove that if $\mathit{ArgLab}'(A) = \texttt{undec}$ then $\mathit{ArgLab}(A) = \texttt{undec}$. Let A be an argument such that $\mathit{ArgLab}'(A) = \texttt{undec}$. Then, by definition of `ConcLab2ArgLab` it holds that:

(1) not for each $v \in \texttt{Vul}(A)$ it holds that $\mathit{ConcLab}(v) = \texttt{out}$, and

(2) there does not exist a $v \in \texttt{Vul}(A)$ such that $\mathit{ConcLab}(v) = \texttt{in}$.

From (1) it follows by definition of `ArgLab2ConcLab` that not for each each argument B with $\texttt{Conc}(B) \in \texttt{Vul}(A)$ it holds that $\mathit{ArgLab}(B) = \texttt{out}$, which implies (Definition 9) that not every attacker of A is labelled `out` by *ArgLab*. Therefore (Definition 11) A is not labelled `in` by *ArgLab*. From (2) it follows by definition of `ArgLab2ConcLab` that there does not exist an argument B with $\texttt{Conc}(B) \in \texttt{Vul}(A)$ and $\mathit{ArgLab}(B) = \texttt{in}$. This implies (Definition 9) that A does not have an attacker that is labelled `in` by *ArgLab*. Therefore (Definition 11) A is not labelled `out` by *ArgLab*. From the thus obtained facts that A is labelled `in` nor `out` by *ArgLab*, it follows that A is labelled `undec` by *ArgLab*. That is, $\mathit{ArgLab}(A) = \texttt{undec}$. □

Lemma 5. *Let* P *be a logic program,* $AF_P = (Ar_P, att_P)$ *be its associated argumentation framework. Let* ArgLab_1 *and* ArgLab_2 *be complete argument labellings of* AF_P*, and* $\mathit{ConcLab}_1$ *and* $\mathit{ConcLab}_2$ *be their respective associated conclusion labellings. It holds that*

1. $\texttt{in}(\mathit{ArgLab}_1) \subseteq \texttt{in}(\mathit{ArgLab}_2)$ *iff* $\texttt{in}(\mathit{ConcLab}_1) \subseteq \texttt{in}(\mathit{ConcLab}_2)$*,*

2. $\texttt{in}(\mathit{ArgLab}_1) = \texttt{in}(\mathit{ArgLab}_2)$ *iff* $\texttt{in}(\mathit{ConcLab}_1) = \texttt{in}(\mathit{ConcLab}_2)$*, and*

3. $\texttt{in}(\mathit{ArgLab}_1) \subsetneq \texttt{in}(\mathit{ArgLab}_2)$ *iff* $\texttt{in}(\mathit{ConcLab}_1) \subsetneq \texttt{in}(\mathit{ConcLab}_2)$*.*

Proof. 1. "$\Rightarrow$": Suppose $\texttt{in}(\mathit{ArgLab}_1) \subseteq \texttt{in}(\mathit{ArgLab}_2)$. Let $c \in \texttt{in}(\mathit{ConcLab}_1)$. Then, by definition of $\texttt{ArgLab2ConcLab}$, there exists an argument $A \in Ar_P$ with $\texttt{Conc}(A) = c$ and $\mathit{ArgLab}_1(A) = \texttt{in}$. From our initial assumption, it follows that $\mathit{ArgLab}_2(A) = \texttt{in}$. So, by definition of $\texttt{ArgLab2ConcLab}$, $c \in \texttt{in}(\mathit{ConcLab}_2)$.
"$\Leftarrow$": Suppose $\texttt{in}(\mathit{ConcLab}_1) \subseteq \texttt{in}(\mathit{ConcLab}_2)$. Let $A \in \texttt{in}(\mathit{ArgLab}_1)$. Then it follows (Definition 11) that each attacker B of A is labelled $\texttt{out}$ by ArgLab_1. That is (Definition 9) for each $B \in Ar_P$ with $\texttt{Conc}(B) \in \texttt{Vul}(A)$ it holds that $\mathit{ArgLab}_1(B) = \texttt{out}$. From the definition of $\texttt{ArgLab2ConcLab}$ it then follows that for each $v \in \texttt{Vul}(A)$ it holds that $\mathit{ConcLab}_1(v) = \texttt{out}$. From our initial assumption it follows (Lemma 3) that $\texttt{out}(\mathit{ConcLab}_1) \subseteq \texttt{out}(\mathit{ConcLab}_2)$. Therefore, $\mathit{ConcLab}_2(v) = \texttt{out}$. This (by definition of $\texttt{ArgLab2ConcLab}$) implies that each argument (say C) with $\texttt{Conc}(C) \in \texttt{Vul}(A)$ is labelled $\texttt{out}$ by ArgLab_2. Therefore (Definition 9) each attacker of A is labelled $\texttt{out}$ by ArgLab_2, so (Lemma 1 A is labelled $\texttt{in}$ by ArgLab_2. That is, $A \in \texttt{in}(\mathit{ArgLab}_2)$.

2. This follows directly from point 1.

3. This follows directly from points 1 and 2.

□

Lemma 6. *Let P be a logic program, $AF_P = (Ar_P, att_P)$ be its associated argumentation framework. Let ArgLab_1 and ArgLab_2 be complete argument labellings of AF_P, and $\mathit{ConcLab}_1$ and $\mathit{ConcLab}_2$ be their respective associated conclusion labellings. It holds that*

1. $\texttt{out}(\mathit{ArgLab}_1) \subseteq \texttt{out}(\mathit{ArgLab}_2)$ *iff* $\texttt{out}(\mathit{ConcLab}_1) \subseteq \texttt{out}(\mathit{ConcLab}_2)$,
2. $\texttt{out}(\mathit{ArgLab}_1) = \texttt{out}(\mathit{ArgLab}_2)$ *iff* $\texttt{out}(\mathit{ConcLab}_1) = \texttt{out}(\mathit{ConcLab}_2)$,
3. $\texttt{out}(\mathit{ArgLab}_1) \subsetneq \texttt{out}(\mathit{ArgLab}_2)$ *iff* $\texttt{out}(\mathit{ConcLab}_1) \subsetneq \texttt{out}(\mathit{ConcLab}_2)$.

Proof. This follows from Lemma 5, together with Lemma 1 and Lemma 3. □

We are now ready to provide some of the key results of the current paper.

Theorem 23. *Let ConcLab be a conclusion labelling of logic program P and associated argumentation framework $AF_P = (Ar, att)$. It holds that ConcLab is a preferred conclusion labelling iff it is a regular conclusion labelling.*

Proof. "$\Rightarrow$": Suppose *ConcLab* is a preferred conclusion labelling. As a direct consequence, there exists a preferred argument labelling *ArgLab* such that $\texttt{ArgLab2ConcLab}(\mathit{ArgLab}) = \mathit{ConcLab}$. The fact that *ArgLab* is a preferred argument labelling means that $\texttt{in}(\mathit{ArgLab})$ is maximal (w.r.t. set-inclusion) among all complete argument labellings. This implies (Lemma 5) that $\texttt{in}(\mathit{ConcLab})$ is maximal (w.r.t. set-inclusion) among all complete conclusion labellings. That is, *ConcLab* is a regular conclusion labelling.
"$\Leftarrow$": Let *ConcLab* be a regular conclusion labelling and take an argument labeling *ArgLab* such that $\texttt{ArgLab2ConcLab}(\mathit{ArgLab}) = \mathit{ConcLab}$. The fact that *ConcLab* is a regular conclusion labelling means that $\texttt{in}(\mathit{ConcLab})$ is maximal (w.r.t. set-inclusion) among all complete conclusion labellings. This implies (by Lemma 5) that $\texttt{in}(\mathit{ArgLab})$ is maximal (w.r.t. set-inclusion) among all complete argument labellings. That is, *ConcLab* is a preferred conclusion labelling. □

Theorem 24. *Let ConcLab be a conclusion labelling of logic program P and associated argumentation framework $AF_P = (Ar, att)$. It holds that ConcLab is the grounded conclusion labelling iff it is the well-founded conclusion labelling.*

Proof. Similar to the proof of Theorem 23. □

Theorem 25. *Let ConcLab be a conclusion labelling of logic program P and associated argumentation framework $AF_P = (Ar, att)$. It holds that ConcLab is an argstable conclusion labelling iff it is a concstable conclusion labelling.*

Proof. "$\Rightarrow$": Let *ConcLab* be an argstable conclusion labelling. This means that there is a stable argument labelling *ArgLab* such that $\texttt{ArgLab2ConcLab}(\mathit{ArgLab}) = \mathit{ConcLab}$. The fact that *ArgLab* is a stable argument labelling means that no argument is labelled `undec`. But then, by Definition 12, also no conclusion in *ConcLab* is labelled `undec`. Hence, *ConcLab* is a concstable conclusion labelling.
"$\Leftarrow$": Let *ConcLab* be a concstable conclusion labelling and take an argument labeling *ArgLab* such that $\texttt{ConcLab2ArgLab}(\mathit{ConcLab})$. The fact that *ConcLab* is a concstable labelling means that no conclusion is labelled `undec` by *ConcLab*. This, by definition of `ConcLab2ArgLab` implies that also no argument in *ArgLab* is labelled `undec`. That is, *ArgLab* is a stable argument labelling, which implies that *ConcLab* is a argstable conclusion labelling. □

In a similar way, one can ask the question of whether semi-stable conclusion labellings are the same as L-stable conclusion labellings. Here, however, the answer is negative. As a counter example, consider the following program:[5]

Example 4. *Consider the following logic program P:*

$$\begin{aligned}
r_1 &: \; c \leftarrow \texttt{not}\ c \\
r_2 &: \; a \leftarrow \texttt{not}\ b \\
r_3 &: \; b \leftarrow \texttt{not}\ a \\
r_4 &: \; c \leftarrow \texttt{not}\ c, \texttt{not}\ a \\
r_5 &: \; g \leftarrow \texttt{not}\ g, \texttt{not}\ b
\end{aligned}$$

One can then build the arguments from P:

- $A_1 = r_1$, *with* $\texttt{Conc}(A_1) = c$ *and* $\texttt{Vul}(A_1) = \{c\}$
- $A_2 = r_2$, *with* $\texttt{Conc}(A_2) = a$ *and* $\texttt{Vul}(A_2) = \{b\}$
- $A_3 = r_3$, *with* $\texttt{Conc}(A_3) = b$ *and* $\texttt{Vul}(A_3) = \{a\}$
- $A_4 = r_4$, *with* $\texttt{Conc}(A_4) = c$ *and* $\texttt{Vul}(A_4) = \{c, a\}$
- $A_5 = r_5$, *with* $\texttt{Conc}(A_5) = g$ *and* $\texttt{Vul}(A_5) = \{g, b\}$

The associated argumentation framework of P is AF_P, depicted in Figure 4.
The complete argument labellings of AF_P are

- $\mathit{ArgLab}_1 = (\emptyset, \emptyset, \{A_1, A_2, A_3, A_4, A_5\})$

[5]We would like to thank Wolfgang Dvořák for this example.

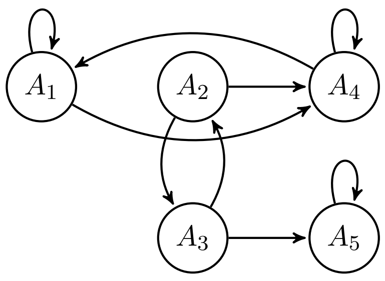


Figure 4: The argumentation framework AF_P associated with P.

- $ArgLab_2 = (\{A_2\}, \{A_3, A_4\}, \{A_1, A_5\})$
- $ArgLab_3 = (\{A_3\}, \{A_2, A_5\}, \{A_1, A_4\})$

The associated complete conclusion labellings are

- $ConcLab_1 = (\emptyset, \emptyset, \{a, b, c, g\})$*,*
- $ConcLab_2 = (\{a\}, \{b\}, \{c, g\})$*, and*
- $ConcLab_3 = (\{b\}, \{a, g\}, \{c\})$*.*

$ArgLab_2$ and $ArgLab_3$ are semi-stable argument labellings. Hence, the associated conclusion labellings $ConcLab_2$ and $ConcLab_3$ are semi-stable conclusion labellings. However, $ConcLab_2$ is not L-stable, because $\texttt{undec}(ConcLab_2)$ *is not minimal. So here we have an example of a logic program where the semi-stable and L-stable conclusion labellings do not coincide.*

We summarize our results concerning the comparison of argument-based conclusion labellings and logic programming-based conclusion labellings in Table 3.

Argument-Based Conclusion Labelling	Relation	Logic Programming-Based Conclusion Labelling
Preferred	$\equiv$	Regular
Grounded	$\equiv$	Well-Founded
Semi-stable	$\not\equiv$	L-stable
Argstable	$\equiv$	Concstable

Table 3: Kinds of conclusion labellings derived from programs

7 On the Connection between Argument Extensions and Logic Programming Models

So far, we have examined the general question of how argument labellings are related to conclusion labellings. We found that:

- maximizing `in` at the argument level yields the same result as maximizing `in` at the conclusion level
- minimizing `in` at the argument level yields the same result as minimizing `in` at the conclusion level
- maximizing `out` at the argument level yields the same result as maximizing `out` at the conclusion level
- minimizing `out` at the argument level yields the same result as minimizing `out` at the conclusion level
- maximizing `undec` at the argument level yields the same result as maximizing `undec` at the conclusion level
- minimizing `undec` at the argument level does *not* yield the same result as minimizing `undec` at the conclusion level

These results are summarized in Table 4.

Cond. on Comp. Arg. Lab.	Name of Ass. Conc. Lab.	Result	Name of Conc. Lab.	Cond. on Comp. Conc. Lab.
None	Complete	[17]	Complete	None
Max. IN	Preferred (Th13)	Th23	Regular (Th17)	Max. IN
Max. OUT	Preferred (Th13)	Th23	Regular (Th17)	Max. OUT
Max. UNDEC	Grd. (Ths 13,16)	Th24	WF (Ths 17,20)	Max. UNDEC
Min. IN	Grd. (Ths 13,16)	Th24	WF (Ths 17,20)	Min. IN
Min. OUT	Grd. (Ths 13,16)	Th24	WF (Ths 17,20)	Min. OUT
Min. UNDEC	Semi-stable	Ex4	L-stable	Min. UNDEC
Empty UNDEC	Argstable	Th25	Concstable	Empty UNDEC

Table 4: Kinds of conclusion labellings derived from programs

Table 4 should be read as follows. The left-hand side of the table is related to the process of maximizing/minimizing a particular labelling at the argument-level, and then generating the associated labellings at the conclusion level (as was outlined in Section 4). Here, we have that selecting the complete argument labellings with maximal `in` is the same as selecting the complete argument labellings with maximal `out` (Th13). These selected argument labellings are called the *preferred argument labellings*, just like their associated conclusion labellings are called the *preferred conclusion labellings.* We also have that selecting the complete argument labellings with maximal `undec` is the same as selecting the complete argument labellings with minimal `in`, and the same as selecting the complete argument labellings with minimal `out` (Ths 13, 16). This unique (Th15) selected argument labelling is called the *grounded argument labelling*, just like its associated conclusion labelling is called the *grounded conclusion labelling* (abbreviated "Grd." in the table).

The right-hand side of the table is related to the process of first generating *all* complete argument labellings and associated complete conclusion labellings, and then to maximize/minimize a particular label at the conclusion level (as was outlined in Section 5). Here, we have that selecting the complete conclusion labellings with maximal `in` is the same as selecting the complete conclusion labellings with maximal `out` (Th23). These selected conclusion labellings are called the *regular conclusion labellings.* We also have that selecting the complete conclusion labellings with maximal `undec` is the same as selecting the complete conclusion labellings with minimal `in`, and the same as selecting the complete conclusion labellings with minimal `out`. This unique (Th19) selected labelling is called the *well-founded conclusion labelling* (abbreviated "WF" in the table).

In the middle column of the table, the connection between these two approaches is indicated. From Th23 it follows that the preferred conclusion labellings are precisely the same as the regular conclusion labellings. From Th24 it follows that the grounded conclusion labelling is precisely the same as the well-founded conclusion labelling. From Th25 it follows that the argstable conclusion labellings are precisely the same as the concstable conclusion labellings. Example 4, however, makes clear that in general, the semi-stable conclusion labellings are *not* the same as the L-stable conclusion labellings. Minimizing `undec` at the argument-level yields fundamentally different results as minimizing `undec` at the conclusion level.

In the current paper, we have proved these results for an instantiation based on logic programs. However, similar results can be obtained also for other forms of instantiated argumentation. In order to apply the proofs that were specified in sections 4, 5 and 6, all that matters is that attack is defined on the conclusion of the attacking argument and the set of vulnerabilities (`Vul`) of the attacked argument. This makes our results directly applicable also to formalisms such as [10] and [3, 18].[6]

7.1 Relating Conclusion Labellings to Models of Programs

In the current paper, we have chosen an instantiation based on logic programming partly because of its relative simplicity, but also because it allows us to study an additional research question: how are (traditional) approaches to argumentation semantics related to (traditional) approaches to logic programming semantics?

In [4] two functions are specified to convert an argument extension to an argument labelling and vice versa. The function `Lab2Ext` converts an argument labelling to a set of arguments (extension) and is defined as $\texttt{Lab2Ext}(\mathit{ArgLab}) = \texttt{in}(\mathit{ArgLab})$. The function `Ext2Lab` converts a conflict-free set of arguments (extension) to an argument labelling and is defined as $\texttt{Ext2Lab}(\mathcal{A}\mathit{rgs}) = (\mathcal{A}\mathit{rgs}, \{A \in Ar \mid \text{there is an argument in } \mathcal{A}\mathit{rgs} \text{ that attacks } A\}, Ar \setminus (\mathcal{A}\mathit{rgs} \cup \{A \in Ar \mid \text{there is an argument in } \mathcal{A}\mathit{rgs} \text{ that attacks } A\}))$. When `Ext2Lab` and `Lab2Ext` are restricted to operate on complete extensions and complete labellings, they become bijective functions that are each other's inverse [4]. This means that complete extensions and complete labellings are one-to-one related, just like preferred extensions and preferred labellings, the grounded extension and the grounded labelling, semi-stable extensions and semi-stable labellings, and stable extensions and stable labellings.

Now that we have observed that the traditional approaches to argumentation semantics (argument extensions) coincide with the approach of argument labellings, the next step is to show that the traditional approaches to logic programming semantics (models based on fixpoints of a reduced program) coincide with the approach of conclusion labellings.

We now introduce two functions `ConcLab2Mod` and `Mod2ConcLab` to convert a conclusion labelling to a model and vice versa. The function `ConcLab2Mod` converts a conclusion labelling to a model and is defined as $\texttt{ConcLab2Mod}(\mathit{ConcLab}) = (\texttt{in}(\mathit{ConcLab}), \texttt{out}(\mathit{ConcLab}))$.

[6]In the latter approaches, the set of vulnerabilities of an argument consists of the statements on which it can be rebutted or undercut.

The function `Mod2ConcLab` converts a model to a conclusion labelling and is defined as $\texttt{Mod2ConcLab}((T, F)) = (T, F, HB_P \setminus (T \cup F))$. It is not difficult to see that `ConcLab2Mod` and `Mod2ConcLab` are bijective functions that are each other's inverse, making conclusion labellings and models one-to-one related.

The next step is to show that 3-valued stable models coincide with complete conclusion labellings. This actually follows from the foundational work of [17]. Here it is proved that if one applies complete semantics at step 2 of the argumentation process, one obtains the 3-valued stable models of the original logic program. From this, together with the results in the current paper, it directly follows that complete conclusion labellings are one-to-one related to 3-valued stable models.

From the correspondence between complete conclusion labellings and 3-valued stable models, the other correspondences between conclusion labellings and models follow. Since a regular model is a 3-valued stable model with maximal T, and a regular conclusion labelling is a complete conclusion labelling with maximal `in`, it follows that they correspond to each other (through the functions `ConcLab2Mod` and `Mod2ConcLab`). Similar correspondences can be observed between the well-founded model and the well-founded conclusion labelling, between L-stable models and the L-stable conclusion labellings and between stable models and concstable conclusion labellings. That is, the various types of logic programming models are actually different forms of conclusion labellings.

7.2 Abstract Argumentation and Logic Programming Semantics Equivalences

We have now arrived at the main point of the current paper: the connection between (traditional) approaches to argumentation semantics and (traditional) approaches to logic programming semantics. Let us again look at the 3-step process of Section 3. Assume that step 1 (AF construction) and step 3 (converting argument labellings to conclusion labellings) are fixed, and that the only degree of freedom is which semantics to apply at step 2. From the results in the current paper, it follows that

- if one applies complete semantics at step 2, the overall outcome is equivalent to calculating the 3-valued stable models to the original logic program[17]
- if one applies preferred semantics at step 2, the overall outcome is equivalent to applying regular semantics to the original logic program
- if one applied grounded semantics at step 2, the overall outcome is equivalent to applying well-founded semantics to the original logic program
- if one applies stable semantics at step 2, the overall outcome is equivalent to applying stable model semantics to the original logic program

That is, differences in logic programming semantics can be reduced purely to differences in what happens at the abstract argumentation level (step 2). In essence, partial stable model semantics coincides with complete semantics, preferred semantics coincides with regular semantics, grounded semantics coincides with well-founded semantics and stable semantics coincides with stable model semantics.

Moreover, we are also able to explain *why* these semantics coincide. Recall that the various argumentation semantics that are studied in the current paper are based on minimization or maximization (of a particular label) at the *argument* level, whereas the various logic programming semantics turn out to be based on minimization and maximization (of a particular label) at the *conclusion* level. The fact that argumentation semantics coincide with logic programming semantics is due to the fact that what happens at the argument level tends to coincide with what happens at the conclusion level. The fact that preferred semantics coincides with regular semantics is *because* maximizing `in` at the argument level is the same as maximizing `in` at the conclusion level. Similarly, the fact that grounded semantics coincides with well-founded semantics is because minimizing `in` at the argument level is the same as minimizing `in` at the conclusion level. Also, the fact that stable semantics coincides with stable model semantics is because ruling out `undec` at the argument level is the same as ruling out `undec` at the conclusion level. Finally, the fact that semi-stable semantics does *not* coincide with L-stable model semantics is because minimizing `undec` at the argument level is something really different from minimizing `undec` at the conclusion level.

8 Semi-Stable Semantics versus L-Stable Semantics Revisited

As we have seen, since minimizing `undec` at the argument-level does not yield the same result as minimizing `undec` at the conclusion level, semi-stable semantics does not coincide with L-stable semantics. In the current section, we study this discrepancy in more detail. In particular, we are interested in the following two questions:

1. is there any abstract argumentation semantics at all that can generate results that are equivalent to L-stable conclusion labellings?
2. is there a class of restricted logic programs for which minimizing `undec` at the argument level yields the same result as minimizing `undec` at the conclusion level?

As for the first question, what we are looking for is an abstract argumentation semantics that can be applied at step 2 of the argumentation process. We assume steps 1 and 3 to remain the same. That is, we are looking for an abstract argumentation semantics that is able to generate the L-stable conclusion labellings, just like preferred semantics is able to generate the regular conclusion labellings and grounded semantics is able to generate the well-founded conclusion labelling. Therefore, the selection of the arguments in such a semantics should be based purely on the *structure* of the graph, and not on the particular contents of the arguments. This can be warranted by requiring that the semantics satisfies the *language independence* principle [1].

Definition 26. *We say that an abstract argumentation semantics X is L-stable generating iff it is a function such that*

1. *For any logic program P, X takes as input AF_P and yields as output a set of argument labellings ArgLabs*
2. *X satisfies language independence, that is, if for any pair of argumentation frameworks AF_1, AF_2, if AF_1 is isomorphic to AF_2 by a mapping M of their arguments (the nodes in the graphs), then each labelling of AF_1 can be mapped to a different labelling of AF_2 by that same mapping M.*

3. It holds that $\{\texttt{ArgLab2ConcLab}(ArgLab) \mid ArgLab \in ArgLabs\}$ *is precisely the same as the set of all L-stable conclusion labellings of* AF_P.

Theorem 27. *There exists no abstract argumentation semantics that is L-stable generating.*

Proof. Consider the following two logic programs P with rules $r_1, ..., r_4$ and P' with rules $r'_1, ..., r'_4$:

$r_1 :$	$c \leftarrow \texttt{not}\ c$	$r'_1 :$	$d \leftarrow \texttt{not}\ c, \texttt{not}\ d$
$r_2 :$	$a \leftarrow \texttt{not}\ b$	$r'_2 :$	$a \leftarrow \texttt{not}\ b$
$r_3 :$	$b \leftarrow \texttt{not}\ a$	$r'_3 :$	$b \leftarrow \texttt{not}\ a$
$r_4 :$	$c \leftarrow \texttt{not}\ c, \texttt{not}\ a$	$r'_4 :$	$c \leftarrow \texttt{not}\ c, \texttt{not}\ a, \texttt{not}\ d$

For the above programs, please observe that:

- P has three 3-valued stable models: $S_1 =< \{\ \}; \{\ \} >$, $S_2 =< \{a\}; \{b\} >$, $S_3 =< \{b\}; \{a\} >$, where S_2, S_3 are L-stable models.
- P' has three 3-valued stable models: $S_1 =< \{\ \}; \{\ \} >$, $S_2 =< \{a\}; \{b, c\} >$, $S_3 =< \{b\}; \{a\} >$, where S_2 is the single L-stable model.

The arguments $A_1, ..., A_4$ built from P and $A'_1, ..., A'_4$ built from P' are listed below.

$A_1 :$	$c \leftarrow \texttt{not}\ c$	$A_{1'} :$	$d \leftarrow \texttt{not}\ c, \texttt{not}\ d$
$A_2 :$	$a \leftarrow \texttt{not}\ b$	$A_{2'} :$	$a \leftarrow \texttt{not}\ b$
$A_3 :$	$b \leftarrow \texttt{not}\ a$	$A_{3'} :$	$b \leftarrow \texttt{not}\ a$
$A_4 :$	$c \leftarrow \texttt{not}\ c, \texttt{not}\ a$	$A_{4'} :$	$c \leftarrow \texttt{not}\ c, \texttt{not}\ a, \texttt{not}\ d$

The argumentation frameworks of P and P' are depicted in Figure 5.

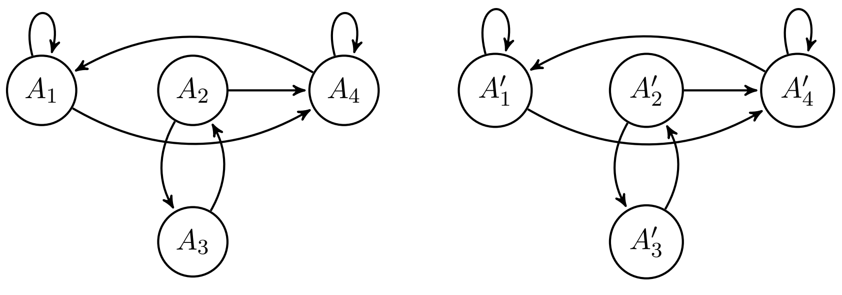


Figure 5: The argumentation frameworks associated with P and P'.

Since P has two L-stable models while P' has only one, the L-stable semantics is sensitive to the difference between them. However, given that these programs have isomorphic associated graphs, they are indiscernible in the perspective of abstract argumentation semantics: By the language independence principle, both argumentation frameworks should have the same number of extensions, but this is not the case. As a consequence, we conclude that no semantics of abstract argumentation can coincide with the L-stable semantics for every program. □

Now that we have observed that in general no abstract argumentation semantics is able to coincide with L-stable model semantics, the next question is whether one can define a restricted class of logic programs for which semi-stable semantics does coincide with L-stable semantics.

Definition 28. *Let P be a logic program. We say that P is* semi-stable-L-stable compatible *iff it holds that* $\{\texttt{ArgLab2ConcLab}(ArgLab) \mid ArgLab$ *is a semi-stable argument labelling of* $AF_P\} = \{ConcLab \mid ConcLab$ *is a L-stable conclusion labelling of P and AF_P*$\}$.

We will now provide two classes of logic programs that are semi-stable-L-stable compatible: The AF-programs and the stratified programs.

Definition 29. *(AF-program) A logic program P is said to be an* AF-program *if (i) every rule r in P has $body^+(r) = \emptyset$; and (ii) there is at most one rule with $head(r) = c$, for each $c \in HB_P$.*

Theorem 30. *Every AF-program is semi-stable-L-stable compatible.*

Proof. Let P be an AF-program and AF_P it's associated argumentation framework. Since each rule has $body^+(r) = \emptyset$, we will have one argument for each rule of P and each such argument will consist of exactly that rule. As a consequence, minimizing undecided arguments is the same as minimizing undecided conclusions and P is semi-stable-L-stable compatible. □

Definition 31. *(stratified program) A logic program P is* stratified *if it is possible to attribute a positive integer $l(p)$ to each atom $p \in HB_P$ in a way that, for each rule r in P with $head(r) = p$, it holds: (i) for each $q \in body^-(r)$, $l(q) < l(p)$; and (ii) for each $q \in body^+(r)$, $l(q) \leq l(p)$.*

Theorem 32. *Every stratified program is semi-stable-L-stable compatible.*

Proof. A stratified program P has a single 3-valued stable model S, so AF_P has a single complete conclusion labelling that coincides with S [17]. Therefore, the only L-stable extension of P coincides with the sole semi-stable conclusion labelling of AF_P. □

Therefore, if P is either an AF-program or an stratified program, the semi-stable conclusion labellings of its associated argumentation framework AF_P are the same as the L-stable models of P.

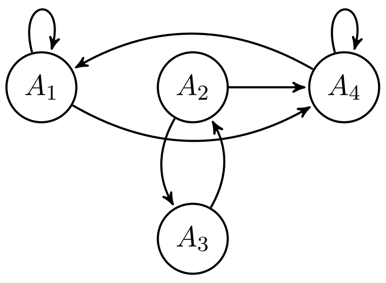
9 Translating Abstract Argumentation Frameworks to Logic Programs

So far, when making the connection between argumentation semantics and logic programming semantics, we have used a translation *from* logic programs *to* argumentation frameworks. In the current section, we will go the other way around. That is, we still examine the connection between argumentation semantics and logic programming semantics, but this time using a translation *from* argumentation frameworks *to* logic programs.

In essence, the idea is to use a translation that is fairly standard in the literature and that has for instance been applied in [17]. Here, each argument generates an associated logic programming rule, with the name of the argument in its head, and the name of its attackers in the weak part of the body.

Definition 33 ([17])**.** *Let $AF = (Ar, att)$ be an argumentation framework. Its associated logic program is* $P_{AF} = \{A \leftarrow \texttt{not}\ B_1, \ldots, \texttt{not}\ B_m \mid A, B_1, \ldots, B_m \in Ar (m \geq 0)$ *and* $\{B_i \mid (B_i, A) \in att\} = \{B_1, \ldots, B_m\}\}$.

Example 5. *Consider AF, the argumentation framework below:*



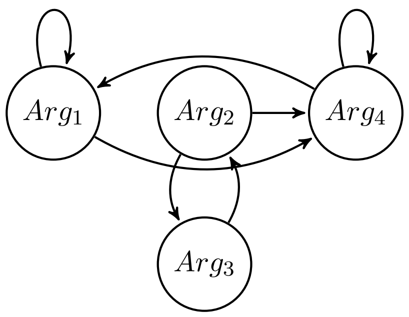
This is the argumentation framework on the left hand side of Figure 5, copied here for easier reference. It's associated logic program P_{AF} is:

$$\begin{array}{ll} r_1: & A_1 \leftarrow \texttt{not}\ A_1, \texttt{not}\ A_4 \\ r_2: & A_2 \leftarrow \texttt{not}\ A_3 \\ r_3: & A_3 \leftarrow \texttt{not}\ A_2 \\ r_4: & A_4 \leftarrow \texttt{not}\ A_4, \texttt{not}\ A_1, \texttt{not}\ A_2 \end{array}$$

The first thing to be observed is that for any argumentation framework AF, the associated logic program P_{AF} is an AF-program (Definition 29).

Now let us examine what happens if we translate P_{AF} back to argumentation theory. That is, what will $AF_{P_{AF}}$ look like? Since each rule in P_{AF} has only a weak part (no strong atoms in the body), each argument will consist of precisely one rule (Definition 8) and the attack relation coincides with the original attack relation in AF.

Example 6. *(Example 5 continued) The argumentation framework $AF_{P_{AF}}$ associated with P_{AF} from Example 5 is:*



Here, each argument Arg_i consists of only the rule r_i from P_{AF}.

Theorem 34. *Let AF be an argumentation framework, P_{AF} be the associated logic program (Definition 33), and $AF_{P_{AF}}$ be the argumentation framework that is associated with this logic program (Definition 10). It holds that AF and $AF_{P_{AF}}$ are isomorphic.*

Proof. Let $AF = (Ar, att)$ and $AF_{P_{AF}} = (Ar_{P_{AF}}, att_{P_{AF}})$. In order to show isomorphism, it suffices to provide a bijective function $f : Ar \to Ar_{P_{AF}}$ such that $(A, B) \in att$ iff $(f(A), f(B)) \in att_{P_{AF}}$. Building such a function is straightforward: We have $f(A_i) = Arg_i$, $i = \{1, 2, 3, 4\}$. That is so because each rule r_i in P_{AF} is built on top of a single $A_i \in Ar$ in a one-to-one relation (a bijection itself) and each $Arg_i \in Ar_{P_{AF}}$ is built on top of a single r_i from P_{AF} (another bijection). $\square$

It has been observed that complete semantics, preferred semantics, grounded semantics, semi-stable semantics and stable semantics all satisfy the language independence principle [1]. This implies that the complete, preferred, grounded, semi-stable and stable labellings of $AF_{P_{AF}}$ are essentially the same (modulo isomorphism) as the complete, preferred, grounded, semi-stable and stable labellings of AF.

What does this mean for the connection between argumentation semantics and logic programming semantics, from the perspective of translating argumentation frameworks to logic programs? First of all, we observe that for the translation of P_{AF} to $AF_{P_{AF}}$ the earlier observed results hold. That is, applying complete semantics to $AF_{P_{AF}}$ yields the same results as applying partial stable model semantics to P_{AF}, applying preferred semantics to $AF_{P_{AF}}$ yields the same results as applying regular semantics P_{AF}, applying grounded semantics to $AF_{P_{AF}}$ yields the same results as applying well-founded semantics to P_{AF}, applying stable semantics to $AF_{P_{AF}}$ yields the same results as applying stable model semantics to P_{AF}, and (due to the fact that P_{AF} is an AF-program) applying semi-stable semantics to $AF_{P_{AF}}$ yields the same results as applying L-stable semantics to P_{AF}.

From the facts that all of the above mentioned argumentation semantics satisfy the language independence principle, and that AF is isomorphic with $AF_{P_{AF}}$, it then follows that applying complete semantics to AF yields the same results as retrieving the 3-valued stable models of P_{AF}, applying preferred semantics to AF yields the same results as applying regular semantics to P_{AF}, applying grounded semantics to AF yields the same results as applying well-founded semantics to P_{AF}, applying semi-stable semantics to AF yields the same results as applying L-stable model semantics to P_{AF} and applying stable semantics to AF yields the same results as applying stable model semantics to P_{AF}. These results have been summarized in Table 5.

Abstract argumentation semantics on AF	Relation	Logic Programming semantics on P_{AF}
Complete	$\equiv$	3-valued Stable
Preferred	$\equiv$	Regular
Grounded	$\equiv$	Well-Founded
Semi-stable	$\equiv$	L-stable
Stable	$\equiv$	Stable

Table 5: The relations between semantics over AF-programs.

Hence, we see that, when translating argumentation frameworks to logic programs, the connection between argumentation semantics and logic programming semantics is stronger than when translating (unrestricted) logic programs to argumentation frameworks. Whereas for the latter translation, semi-stable and L-stable do not coincide, for the former translation they do.

10 Discussion

In this paper we studied various connections amongst abstract argumentation semantics and logic programming semantics. We started by giving proper definitions of these in a way that already suggests their connections. In the same spirit, we introduced a division operation over argumentation frameworks which resembles the Gelfond-Lifschitz program transformation from [9]. Based on that newly introduced concept, we provided an alternative for the definition of argumentation semantics. After these definitions, we would start looking deeper into the reasons why the semantics coincidence. By presenting a three-step process of instantiated argumentation, we observed that differences in step 2 (applying abstract argumentation semantics) are able to fully account for the differences between various logic programming semantics. In the then following sections, we examined how the argumentations labellings and conclusion labellings of argumentation frameworks relate to each other and how these, in their turn, relate to the models from the various logic programming semantics. In doing so, we showed how logic programming semantics is intimately connected to the set of arguments that can be built from a logic program. This is an important result, especially because the equivalences between argument labellings and conclusion labellings also apply to other instantiations of abstract argumentation, like ASPIC [3] and ASPIC+ [11].

Another important aspect of our contributions concerns instantiated argumentation in general, as we showed how arguments at the abstract level relate to conclusions at the instantiated level of argumentation: Our results show that, in general, analyzing the attack relation amongst arguments is enough to retrieve the possible sets of conclusions from such arguments. We analyzed various ways of distinguishing admissible sets of arguments and conclusions, such as maximizing or minimizing those arguments (or conclusions) that are accepted, rejected, or undecided. In that context, we provided several equivalence results and brought the sole exception to attention, namely the situation when undecided arguments are minimized. This result was shown for instantiated argumentation in general, i.e., it was shown that choosing extensions where the set of arguments left undecided is minimal is not the same as minimizing undecidedness on the possible conclusions. In the particular context of logic programs, minimizing undecided conclusions characterizes the L-stable models. On that matter, we proved that abstract argumentation semantics cannot capture the L-stable semantics for normal logic programs. We then showed two classes of programs to which the semi-stable argumentation semantics is capable capture to capture the L-stable models, namely the here defined AF-programs and the well-known class of stratified programs.

Finally, we analyzed the semantics connections the other way around, i.e., by associating a logic program to an argumentation framework. In that sense, we showed how to properly build a program from an argumentation framework in such a way that the models from each logic programming semantics corresponds exactly the extensions of an argumentation semantics. The connection amongst argumentation and logic programming semantics turns out to be even stronger when translating argumentation frameworks to logic programs, then when translating (unrestricted) logic programs to argumentation frameworks.

Acknowledgements

This work has been supported by the National Research Fund, Luxembourg (LAAMI project) and by the Engineering and Physical Sciences Research Council (EPSRC, UK), grant ref. EP/J012084/1 (SAsSy project). We would also like to thank Wolfgang Dvořák for his useful comments.

References

[1] P. Baroni, M.W.A. Caminada, and M. Giacomin. An introduction to argumentation semantics. *Knowledge Engineering Review*, 26(4):365–410, 2011.

[2] M.W.A. Caminada. On the issue of reinstatement in argumentation. In M. Fischer, W. van der Hoek, B. Konev, and A. Lisitsa, editors, *Logics in Artificial Intelligence; 10th European Conference, JELIA 2006*, pages 111–123. Springer, 2006. LNAI 4160.

[3] M.W.A. Caminada and L. Amgoud. On the evaluation of argumentation formalisms. *Artificial Intelligence*, 171(5-6):286–310, 2007.

[4] M.W.A. Caminada and D.M. Gabbay. A logical account of formal argumentation. *Studia Logica*, 93(2-3):109–145, 2009. Special issue: new ideas in argumentation theory.

[5] M.W.A. Caminada and G. Pigozzi. On judgment aggregation in abstract argumentation. *Autonomous Agents and Multi-Agent Systems*, 22(1):64–102, 2011.

[6] M.W.A. Caminada and B. Verheij. On the existence of semi-stable extensions. In G. Danoy, M. Seredynski, R. Booth, B. Gateau, I. Jars, and D. Khadraoui, editors, *Proceedings of the 22nd Benelux Conference on Artificial Intelligence*, 2010.

[7] P.M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77:321–357, 1995.

[8] Th. Eiter, N. Leone, and D. Saccá. On the partial semantics for disjunctive deductive databases. *Ann. Math. Artif. Intell.*, 19(1-2):59–96, 1997.

[9] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R.A. Kowalski and K. Bowen, editors, *Proceedings of the 5th International Conference/Symposium on Logic Programming*, pages 1070–1080. MIT Press, 1988.

[10] N. Gorogiannis and A. Hunter. Instantiating abstract argumentation with classical logic arguments: Postulates and properties. *Artificial Intelligence*, 175(9-10):1479–1497, 2011.

[11] H. Prakken. An abstract framework for argumentation with structured arguments. *Argument and Computation*, 1(2):93–124, 2010.

[12] H. Prakken and G. Sartor. Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-Classical Logics*, 7:25–75, 1997.

[13] T.C. Przymusinski. The well-founded semantics coincides with the three-valued stable semantics. *Fundamenta Informaticae*, 13(4):445–463, 1990.

[14] G.R. Simari and R.P. Loui. A mathematical treatment of defeasible reasoning and its implementation. *Artificial Intelligence*, 53:125–157, 1992.

[15] Emil Weydert. Semi-stable extensions for infinite frameworks. In Patrick de Causmaecker, Joris Maervoet, Tommy Messelis, Katja Verbeeck, and Tim Vermeulen, editors, *Proceedings of the 23rd Benelux Conference on Artificial Intelligence (BNAIC 2011)*, pages 336–343, 2011.

[16] Y. Wu and M.W.A. Caminada. A labelling-based justification status of arguments. *Studies in Logic*, 3(4):12–29, 2010.

[17] Y. Wu, M.W.A. Caminada, and D.M. Gabbay. Complete extensions in argumentation coincide with 3-valued stable models in logic programming. *Studia Logica*, 93(1-2):383–403, 2009. Special issue: new ideas in argumentation theory.

[18] Yining Wu. On the issue of non-interference in the aspic-light formalism. Technical report, University of Luxembourg, 2011.